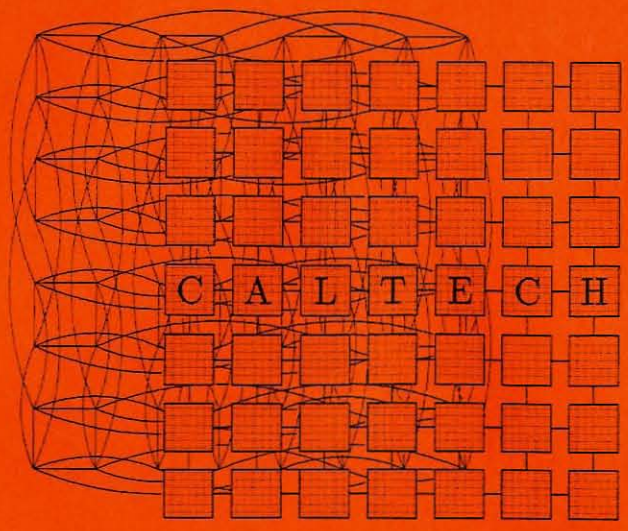


SUBMICRON SYSTEMS ARCHITECTURE PROJECT
Department of Computer Science
California Institute of Technology
Pasadena, CA 91125

Submicron Systems Architecture Project

Semiannual Technical Report

Caltech Computer Science Technical Report

Caltech-CS-TR-92-17

1 July 1992

The research described in this report was sponsored by
the Defense Advanced Research Projects Agency
and monitored by the Office of Naval Research.

SUBMICRON SYSTEMS ARCHITECTURE

Semiannual Technical Report

Department of Computer Science
California Institute of Technology

Caltech-CS-TR-92-17

1 July 1992

Reporting Period:	1 November 1991 — 30 June 1992 (8 months)
Principal Investigator:	Charles L. Seitz
Faculty Investigators:	Alain J. Martin Charles L. Seitz Jan L. A. van de Snepscheut

Sponsored by the
Defense Advanced Research Projects Agency

Monitored by the
Office of Naval Research

SUBMICRON SYSTEMS ARCHITECTURE

Department of Computer Science
California Institute of Technology

1. Overview and Summary

1.1 Scope of this Report

This report is a *summary* of research activities and results for the eight-month period, 1 November 1991 to 30 June 1992, under the Defense Advanced Research Projects Agency (DARPA) Submicron Systems Architecture Project. Previous semiannual technical reports and other technical reports covering parts of the project in detail are listed following these summaries, and can be ordered from the Caltech Computer Science Library.

1.2 Objectives

The central theme of this research is the architecture and design of VLSI systems appropriate to a microcircuit technology scaled to submicron feature sizes. Our work is focused on VLSI architecture experiments that involve the design, construction, programming, and use of experimental multicomputers (message-passing concurrent computers), and includes related efforts in concurrent computation and VLSI design.

1.3 Highlights

- A 128-node Mosaic C multicomputer was constructed from two 8×8 boards, and its correct operation has been verified both with application and stress-test programs. (section 2.1).
- The C+− programming system (section 2.2) is operational, and was used to write the Mosaic C runtime system (section 2.3).
- The Affinity (formerly called the Page Kernel) concurrent-programming system was completed, demonstrated, and documented (section 3.1).
- Several application programs written using the Mosaic Pascal programming system have been developed and benchmarked (sections 3.2–3.4).
- Simulation studies have yielded new results in understanding the performance multicomputer message-passing networks (section 4.1).
- A family of high-speed routing, communication, and interface chips is being developed for commercial products and for other research projects (section 4.2).
- The Asynchronous Microprocessor was successfully ported to GaAs (section 4.3).
- New results in testing asynchronous logic (section 4.4).

2. The Mosaic Project

Background and Summary. The Mosaic C is an experimental *fine-grain multicomputer* based on single-chip nodes. The Mosaic C chip includes 64KB of fast dynamic RAM, processor, packet interface, ROM for bootstrap and self-test, and a two-dimensional self-timed router. The chip architecture provides low-overhead and low-latency handling of message packets, and high memory and network bandwidth. Sixty-four Mosaic chips are packaged by tape-automated bonding (TAB) in an 8×8 array on circuit boards that can, in turn, be arrayed in two dimensions to build arbitrarily large machines. These 8×8 boards are now in prototype production under a subcontract with Hewlett-Packard. We are planning to construct a 16K-node Mosaic C system from 256 of these boards. The suite of Mosaic C hardware also includes host-interface boards and high-speed communication cables. The hardware developments and activities of the past eight months are described in section 2.1.

The programming system that we are developing for the Mosaic C is based on the same message-passing, reactive-process, computational model that we have used with earlier multicomputers, but the model is implemented for the Mosaic in a way that supports fine-grain concurrency. A process executes only in response to receiving a message, and may in execution send messages, create new processes, and modify its persistent variables before it either exits or becomes dormant in preparation for receiving another message. These computations are expressed in an object-oriented programming notation, a derivative of C++ called C+−. The computational model and the C+− programming notation are described in section 2.2. The Mosaic C runtime system, which is written in C+−, provides automatic process placement and highly distributed management of system resources. The Mosaic C runtime system is described in section 2.3.

2.1 Mosaic C hardware

2.1.1 The Mosaic C chip

Chuck Seitz, Jakov Seizovic, Wen-King Su

A plot of the production version (M1.2) of the Mosaic C chip is shown in figure 1. The bounding-box dimensions are 9.25mm×10.00mm, the transistor count is ≈1.2M including the 512K transistors used as storage capacitors in the dynamic RAM, and the chip is fabricated in 1.2μm CMOS technology (MOSIS SCMOS with $\lambda = 0.6\mu$m (n-well), or the Hewlett-Packard CMOS34 process). The chip operates with large margins at a 30MHz clock rate, and dissipates ≈0.5W. The lower 3/4 of the chip is the 64KB of fast dynamic RAM, which is organized as 32K 16-bit words. Across the top 1/4 of the chip, left to right, is the processor and packet interface, clock driver, ROM for bootstrap and self-test, and two-dimensional self-timed router. The 136 pins are devoted principally to the four bidirectional channels that connect to the north, south, east, and west neighbors. The pins also include timing inputs and amplified outputs to distribute clock, reset, and refresh signals through the mesh; LED and tachometer outputs; and 36 Vdd and GND pins.

A detailed, chronological report of our experiences during this reporting period may be useful to others who may be preparing prototype chips for production.

The Mosaic C 1.0 chip (M1.0) was fabricated through MOSIS in June–August 1991, and we were able to report at last fall's VLSI contractor's meeting that this chip worked correctly on first silicon. Small Mosaic systems with M1.0 chips in PGA packages on 3×3 and 4×4 boards allowed us to verify the design even to the level of running application programs.

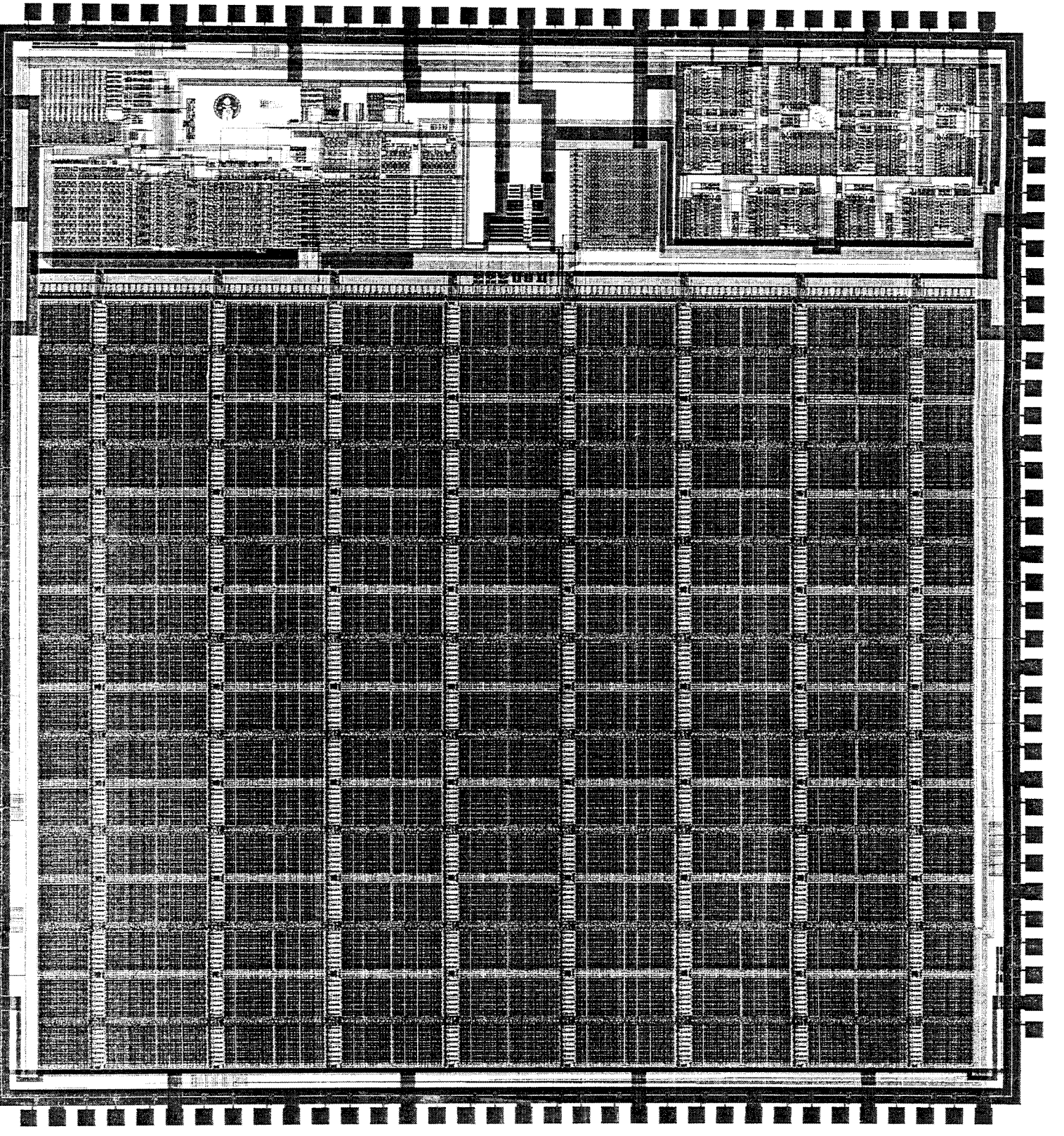


Figure 1: Plot of the Mosaic C 1.2 chip.

The M1.1 is the same as the M1.0 except for correcting a clock-amplifier inversion mistake. A lot of 48 wafers was fabricated directly through HP, with MOSIS performing the data conversion. These chips came out of fabrication on 10 December 1991, and were tested on the wafers. Wafer testing did not present any unusual problems; we had good, initial, wafer-test results by 13 December. By bonding and testing chips from four of the tested wafers, we found subsequently that the wafer-test program rejected certain chips due to one overly restrictive timing requirement. After correcting the wafer-test program, the yield on this M1.1 run came out to be slightly better than 50%. The yield determines the cost per working chip. At 52 total die and 27 functional die per \$800 (tested, 4″) wafer, the unpackaged silicon cost works out to be about \$30 per working chip, or just under \$2000 per 8×8 board.

In addition to being the vehicle for developing the wafer-test programs, these M1.1 chips were, as planned, used to develop the inner-lead bonding to the TAB frames, the encapsulation, the functional test in the TAB frame, the outer-lead bonding to the 8×8 boards, and the board-test programs. The M1.1 was also subjected to HP's full battery of geometrical-design-rule (GDR) checks, and, as it turned out, was used to verify the electrostatic-discharge (ESD) and latchup-protection characteristics of the pad circuits.

There were still problems with the M1.1 chips that we either needed or wanted to fix before putting the Mosaic into production. The most serious problem was the router, which, under certain conditions, exhibited the same types of errors that have been observed at very low rates with the Caltech FMRC (Frontier) router in the Intel Delta, but at higher rates in the Mosaic due to its ability to generate much higher volumes of message traffic. Although it didn't seem to be hurting the yield, both the processor and the router contained small GDR errors due to changes made to the SCMOS GDRs after these sections of the Mosaic chip were designed. Between 13 December 1991 and 15 January 1992, Chuck Seitz and Wen-King Su designed and laid out a new router, designated as the "Elko" router, and Jakov Seizovic fixed all of the GDR errors in the processor. Since the router needed to operate at only 60MB/s to keep up with the packet interface operating at a 30MHz word rate, we were able to produce a conservative, must-work-first-time design of the Elko routing automata.

This intense five weeks resulted in (1) an M1.2 layout, (2) a memoryless Mosaic 3.7 (MM3.7) that contained the new processor and Elko router, and (3) an Elko mesh-routing chip (EMRC1.0) that is pin-compatible with the FMRC2.3 router used in the Intel Delta. The M1.2 layout was converted from CIF to MEBES by MOSIS, and was sent to Hewlett-Packard. The MM3.7 and EMRC1.0 were submitted to MOSIS for the 1.2μm run that closed 15 January 1992.

Hewlett-Packard subjected the M1.2 layout data to so many checks to qualify the chip for HP production that we decided to delay the M1.2 fabrication until we received and could test the new processor and router in the MM3.7 and EMRC1.0 chips. Due to a rather acerb review of our pad-protection structures (the HP engineer obviously didn't understand them, other than that they were different from HP pads), we suggested that the characteristics of these pads be verified by electrical testing of M1.1 chips. The M1.1 chips passed HP's ESD tests at a higher voltage than production HP chips, but HP was able to induce a latchup in the Schmitt-trigger pads used for the request and acknowledge signals. We redesigned this input circuitry to eliminate the latchup. We also made 12 small changes to the M1.2 layout in response to HP's excellent GDR-checking program; these were not GDR violations, but improvements to plugging wells. The only known shortcoming in the current M1.2 layout is a myriad of n-select, p-select overlaps that Magic produces using the current SCMOS

technology file. These overlaps are harmless since they don't occur over active, but HP runs a program to eliminate them from the pattern files.

We received the new Elko router from MOSIS on Friday 6 March 1992, and the memoryless Mosaic 3.7 on Monday 9 March. Both chips worked perfectly in the extensive tests we subjected them to during the week of 9 March. In spite of the elimination of the small GDR violations, there was no significant difference in the yield between these routers and memoryless Mosaics chips and their predecessors.

We were, accordingly, ready to proceed with the M1.2. We sent the revised M1.2 CIF to MOSIS on Monday evening 16 March, and Wes Hansford sent the MEBES tape to Hewlett-Packard on Tuesday afternoon 17 March. Fabrication was completed by 21 April, and, after correcting a small testing snafu, the yield on this 48-wafer prototype-production run came out to be 31%, with an unusually large variation in yield from wafer to wafer. Ironically, fixing the GDR violations did not improve the yield, at least not on this run.

Bottom Line: With only minor changes, this MOSIS-SCMOS chip laid out with Magic became fully qualified for HP *production* without having to ask HP to waive anything.

2.1.2 Mosaic 8×8 boards

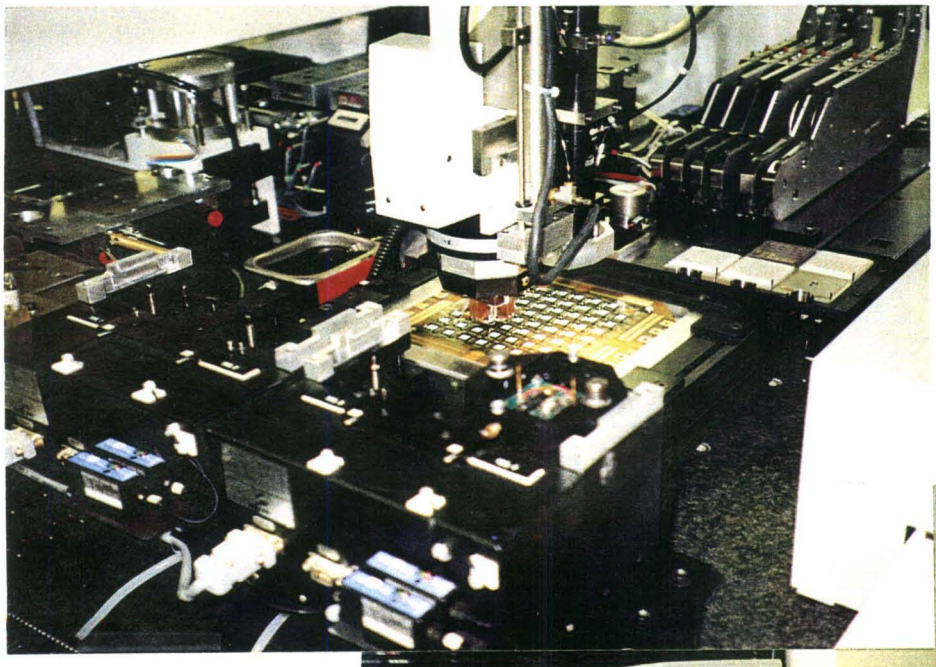
Chuck Seitz, Wen-King Su

In contrast with our earlier system-building experiments, the Mosaic C is a "high-tech" machine composed nearly entirely of custom silicon of our own design. In order to demonstrate how well arrays of mesh-connected chips could be packaged and tested, and to achieve economies in building a 16K-node prototype, we decided to package the Mosaic chips in a way that lends itself to volume production: tape-automated bonding (TAB) on circuit boards. This type of packaging requires a substantial amount of custom tooling. We were able to negotiate an agreement with Hewlett-Packard to develop the TAB tooling and board layout at their cost, in part because Hewlett-Packard wished to use the Mosaic C as a vehicle for developing, in conjunction with MCC, a 10mil-outer-lead TAB process. The Mosaic chips also employ 10mil-inner-lead spacing, so that the TAB structure is "direct," *eg,* it does not include a fan structure to change pitch between the inner and outer leads. The result is that the Mosaic chips can be arrayed onto circuit boards at a density similar to that of multi-chip modules. In this mesh-interconnect structure, the high-speed signals between the routers are all conveyed on short wires that exhibit very low capacitance and inductance.

Figure 2, a set of photographs taken at MCC, illustrate parts of the assembly and test processes for these "high-tech" circuit boards. The complete manufacturing flow consists of:

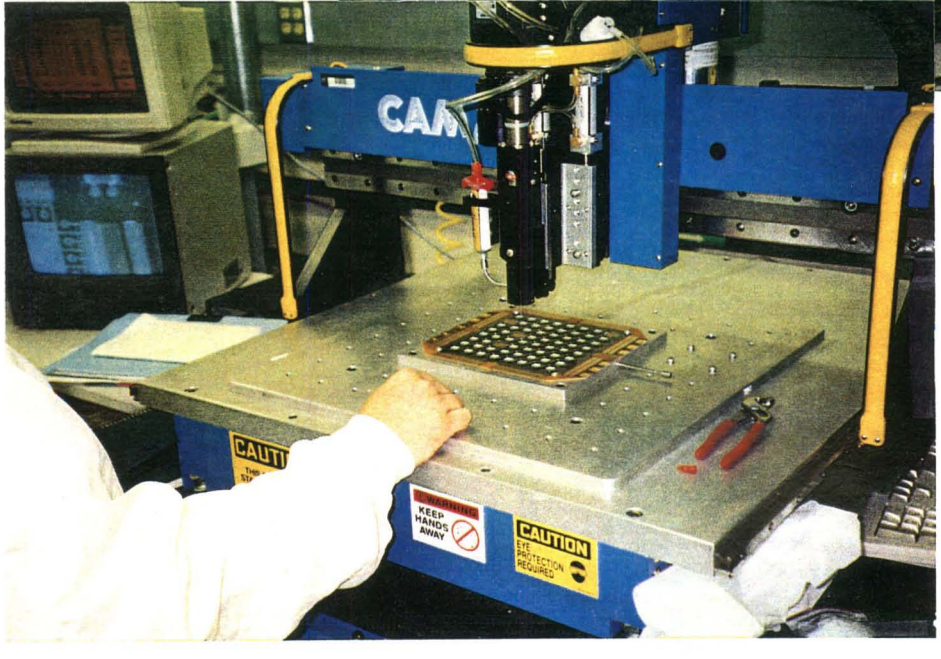
- wafer fabrication,
- wafer test,
- *inner-lead* TAB bonding and chip encapsulation,
- retest in the TAB frame,
- outer-lead bonding onto the circuit boards,
- diagnostic board test, and
- possible repair of the board until it passes the board test.

In order to automate the testing of 8×8 boards at the point of manufacture, a test fixture interfaces to two corner nodes, (0,0) and (7,7), through standard Mosaic channels. These corner channels are supplied by a Mosaic Sbus host-interface board in a Sun SPARCstation.



TAB Bonding

Testing



Reworking

Figure 2: Assembly and testing of the Mosaic 8×8 boards at MCC.

All other edge channels are looped back to allow the edge channels to be tested. An X-window-based, turn-key, test program was written to be used in conjunction with this test fixture, and the entire setup was delivered to and installed at MCC in May 1992. The test system is currently in use for screening 8×8 boards built with M1.2 chips.

After a board to be tested is clamped down in the test fixture, testing begins with the operator running the test program on the SPARCstation. The program displays an image of the board under test, and on that image it overlays a dynamic display of the testing activity and partial results. The program begins by downloading test programs into the nodes at the two corners, and proceeds by using nodes that are found to be functional to test neighboring nodes and the channels that connect them. After the test is completed, the board is either completely functional, or the operator can decide based on the display which chips are either defective or were improperly bonded.

Figure 3 is a photograph of an 8×8 board in operation in our laboratory.

2.1.3 Mosaic Sbus host-interface boards

Wen-King Su

Over the past two years, VME-based 4-node Mosaic host-interface boards have been very successful as platforms for Mosaic experiments ranging from program development to their use in the ATOMIC LAN. However, the changing mix of the project's computing equipment from VME-based Sun-3s to Sbus-based SPARCstations, together with the desire for higher bandwidth between hosts and Mosaics, persuaded us to develop an Sbus-based Mosaic host-interface board. Figure 4 is a photograph of the 2-node Mosaic Sbus host-interface board in a SPARCstation. The choice of a 2-node configuration is due to the small, postcard size of the Sbus board, $3.5'' \times 5.75''$, but multiple Sbus interfaces can be chained in a single SPARCstation if desired.

Other than an extra ROM chip and a different control-register layout to meet the SBus specifications, the new Sbus host-interface board is essentially the same as a 2-node version of the VME-based host-interface board. The board contains 2 memoryless Mosaic chips, 4 SRAM chips for each memoryless Mosaic to provide the 64K×16 memory, and the control to allow both the Sbus and the memoryless Mosaics to access the memories. The device driver also mirrors that of the VME-based interface board, such that applications need only recompile to make use of the Sbus board.

Although the Sbus boards are rated at 25MHz, most existing SPARCstations operate the bus at 20MHz. The host-interface boards will operate correctly at up to a ≈38MHz clock rate with 15ns SRAM chips, but, to simplify the synchronization, are limited on the Sbus boards to operating at the Sbus clock rate. However, because the Sbus hardware automatically maps a 32-bit access into two consecutive 16-bit accesses, and because it takes fewer clock cycles for an access, the Sbus host-interface boards have a higher host↔interface transfer rate than the VME-based host-interface boards. A total of 10 boards have now been built, and all of them are currently in use.

The latest iterations of the memoryless Mosaic chips, MM3.7y and MM3.8y, have been fabricated in a new "y" pinout. In order to confine the number of pins in the memoryless Mosaic chips, only a subset of the routing channels is brought out to the pad frame; the other pins are used for the interface to the external memory and Sbus. The older memoryless Mosaics used the subset consisting of east-in, east-out, west-in, west-out, and south-in. This subset allows bidirectional x chaining together with a y input to receive packets from any node in an array in which packets are routed in x-first-then-y order. Based on a suggestion

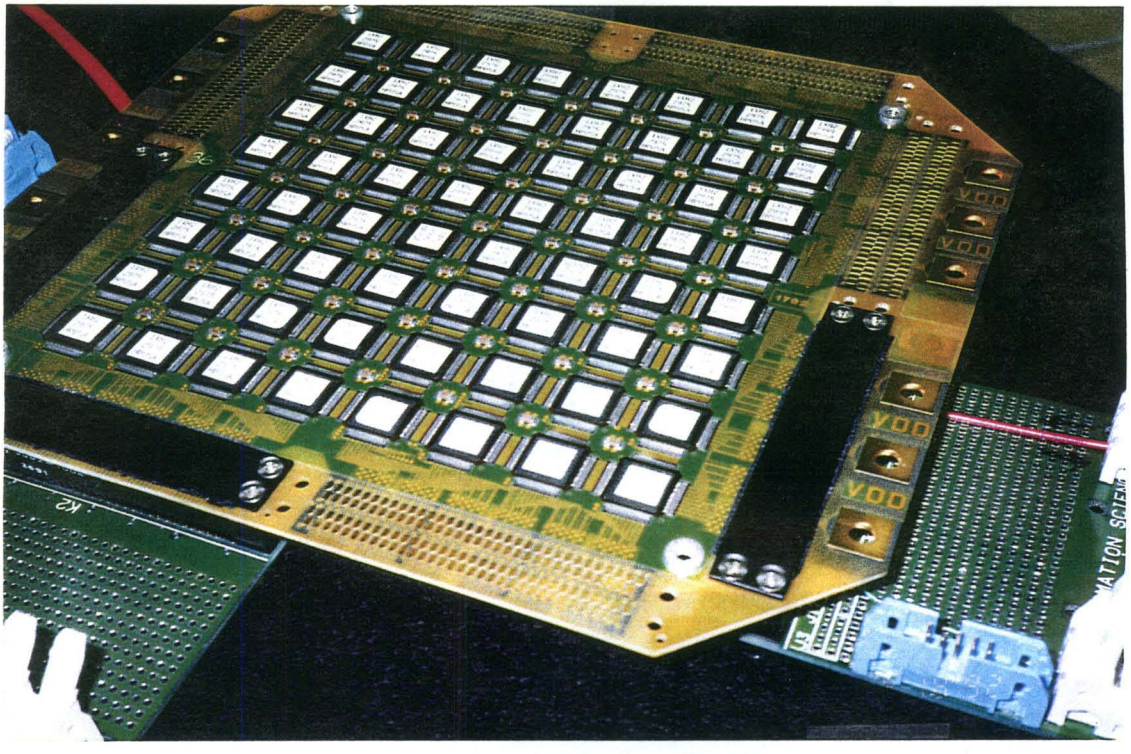


Figure 3: A fully operational Mosaic 8×8 board connected to cable-adapter boards.

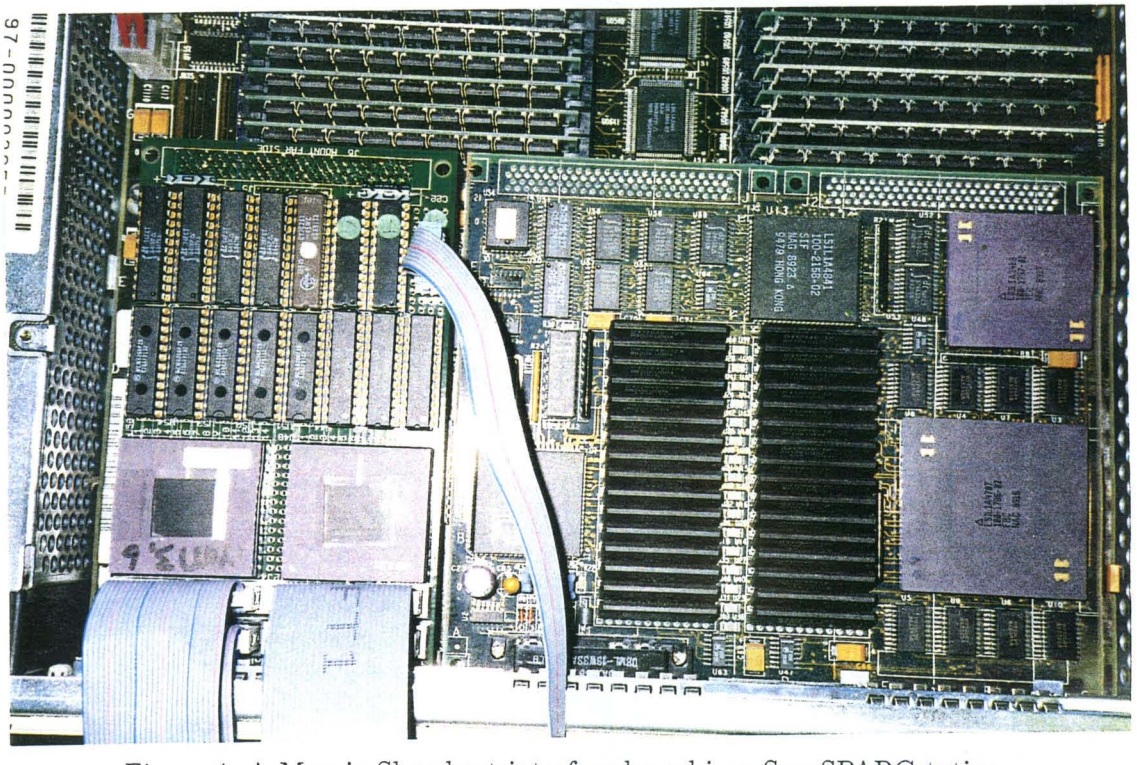


Figure 4: A Mosaic Sbus host-interface board in a Sun SPARCstation.

from members of the ATOMIC project at USC/ISI, we have now standardized on the subset: east-in, east-out, north-in, and north-out. This subset still allows chaining, as illustrated in figure 5; in addition, any host-interface node can route packets directly to any array node, and any array node can route packets directly to any host-interface node.

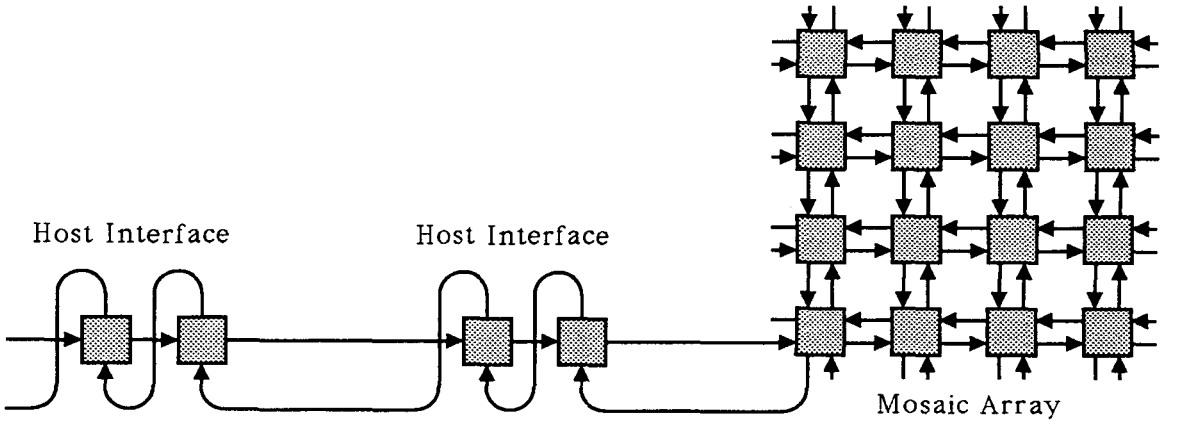


Figure 5: Standard connection of Mosaic host-interfaces and arrays.

Figure 5 also illustrates that, in addition to the Mosaic host-interface and array boards, complete Mosaic systems require cables that connect between host interfaces and arrays. The zero-slack (non-interference) protocol used between Mosaic nodes that are physically close also operates correctly over long cables, but the bandwidth is limited by the round-trip cable delay. Slack chips that convert between zero-slack and slack protocols have been developed, and are described in section 4.2. These slack chips will be used together with transmission-line drivers and receivers at the ends of cables up to $\approx 50'$ in length, and will allow the same 60MB/s bandwidth on long cables as is provided between Mosaic nodes that are physically close.

2.1.4 Programming Toolkit

Wen-King Su

The low-level programming toolkit for the Mosaic includes a gcc-based C compiler, a library of communication routines, example programs, test programs, and device drivers for the VME and Sbus host-interface boards. Only a few changes to this toolkit were required over the past eight months. However, the C compiler, currently based on gcc 1.4, will soon be upgraded to gcc 2.0 in order to be able to use new features of g++ that are available only with gcc 2.0.

2.1.5 Retrospective

Chuck Seitz

The Mosaic multicomputer was the product of several years of design effort by many talented people. In addition to its value as an architectural experiment, the development of the Mosaic hardware has required, stimulated, and demonstrated innovations in VLSI design, automated testing, packaging, and prototyping, including particularly:

- A dense, fast, dynamic RAM that can be fabricated with the MOSIS SCMOS process.
- High-performance cut-through routers and communication chips.
- A multisequence processor that can switch between message-handling and user contexts in zero time.
- A pipeline synchronizer between the asynchronous router and synchronous memory. The technique used in this pipeline synchronizer is an elegant solution to a 20-year-standing problem of how to perform reliable synchronization on data streams whose throughput is higher than reliable synchronization rates.
- An elegant demonstration of the use of MOSIS prototyping services to build, test, and use functional modules, such as the memoryless-Mosaic-based host-interface boards and a variety of routing chips, in advance of investing in "high-tech" packaging for scaling up to large systems.
- A demonstration of the density and economy of TAB-on-circuit-board packaging for mesh-connected systems.
- A demonstration of the effectiveness of ROM self-test, and of automatic testing and diagnosis of faults in mesh-connected structures.
- An early technology-transfer demonstration in which the modules and programming toolkit developed for this scalable, high-performance, fine-grain multicomputer were employed during its development in an embedded-system application, the ATOMIC LAN.

The hardware-development aspects of the Mosaic C project are now essentially complete. A 128-node Mosaic C system built from two HP-manufactured and -tested 8×8 boards, and connected to Sbus host-interface boards in a SPARCstation, is able to run application programs and our most demanding stress-test programs. This demonstration, with the 8×8 boards tiled either horizontally or vertically, verifies both the chip and board designs.

The Mosaic C components are also manufacturable. The past eight months of the Mosaic project encompassed the substantial effort that is known to separate the "Look! It works!" stage of a project from the ability to scale up economically to large systems.

We have been pleased with our relationship with Hewlett-Packard. As with any complex engineering project, some give-and-take has been necessary. Both we and HP have had to do some things we didn't expect, but we have each taken on these extra tasks according to what we each knew how to do best. We have been impressed with the quality of their work, whether an extra computer check of our layouts or a test fixture for the 8×8 boards. They have similarly been pleased, and probably pleasantly surprised, for example, that our chip designs, chip-test programs, and board-test programs are so polished – something we were able to do only because of prototype chips fabricated through MOSIS; and because of system prototypes such as our host-interface, 3×3, and 4×4 boards, which were built with

MOSIS chips. It has certainly also helped that the HP people are enthusiastic about this project.

The MOSIS crew, particularly Wes Hansford and Sam Reynolds, have been very helpful in doing the CIF→CMOS34 conversions for us, and accommodating us on regular MOSIS runs.

2.2 The C+− Programming System

Jakov Seizovic, Chuck Seitz

C+− is an effort to achieve the advantages of a compilation-based programming system and of object-oriented programming in expressing and executing message-passing, reactive-process, concurrent programs. The C+− experiment consists of:

- Defining the programming model;
- Mapping the programming model into a notation, in our case, into C++ with a handful of extensions;
- Developing a C+− to C++ translator;
- Supporting C+−'s use as the source notation for a run-time system for the Mosaic C (see section 2.3) and for other multicomputers (including networks of workstations); and
- Writing application programs that test the expressivity of C+− and the efficiency of the implementation.

C+− has been strongly influenced by our group's earlier efforts in multicomputer programming systems. The capabilities of C+− are essentially similar to those of the widely-used Cosmic Environment / Reactive Kernel (CE/RK, or Cosmic C) system and of Reactive C. C+−, however, allows the source program to be expressed lexically together, such that C+− can perform checks at compile time that, in these earlier systems, were performed at run time, if at all. The Cantor programming system, our group's first compilation-based concurrent-programming system, is also part of C+−'s genealogy; however, C+− supports the full range of object-oriented features, provides for message discretion, and, because it is based on C++, is a rich rather than a minimalist programming environment. The Mosaic C has also been a driving force and reality test behind this effort. Design decisions have consistently been made in a way that avoided compromising the performance of C+− programs on the Mosaic.

Programming Model. We are witnessing a proliferation of programming systems that attempt to merge particular object-oriented languages and concurrent semantics. Almost all of these systems are motivated primarily by networks of workstations, for which the message latency is in the order of milliseconds. This target favors client-server models, and leads their designers to Remote Procedure Call (RPC) semantics. Concurrency is generally introduced after-the-fact, typically by using futures (synchronization variables).

In our world of fine-grain multicomputers, the message latency is only a few microseconds. Both in our work on VLSI design and in the design and implementation of programming and operating systems for concurrent computers, performance considerations provide strong arguments for a lowest-level model that is more streamlined than RPC. For nearly a decade, our research group has successfully been using reactive-process semantics in which concurrent processes exchange messages. RPC semantics can be implemented *where necessary* in terms of these simpler semantic elements. We regard this approach as

"successful" in that its expressivity has been demonstrated by thousands of application programs, and its highly streamlined message handling allows high efficiency. C+−'s only departure from the programming model used in our earlier systems is that, instead of a single entry point (also characteristic of Actors), processes have a fixed, compile-time-determined set of entry points, each of which can be enabled or disabled at run time. The effect of this slight departure is to restrict and streamline the dispatch of messages to the functions that process them.

Programming Notation. The C++ class concept is carried over intact: `class` is a user-defined type; an *object* created from a class definition is a piece of memory with a set of operations defined on it, and a set of access rules. Operations on objects have instantaneous action, characteristic of sequential programming models. C+− introduces the *process* concept: `processdef` parallels the `class` keyword syntactically, but a *process* created from a `processdef` represents a user-defined virtual machine. A process is a computing agent, and a unit of potential concurrency. Its alphabet is defined by the set of entry points, and its operations have delayed action. This delayed action is characteristic of concurrent programming models that expose the inevitable latency of communication, rather than attempting to hide latency from (some) users.

C+− Translator. The C+− to C++ translator's main concerns are global name-space management (process pointers can be dereferenced globally), and optimizations (stack *versus* message-space management, implementation of RPC in terms of simpler communication primitives, and support for communicating arbitrary data structures).

Preliminary Evaluation. Although the experiment is still in progress, C+− has been able to support both a runtime system (see section 2.3) and user-application programming with almost no compromises in either.

- *Efficiency:* C+− imposes a very low software-overhead penalty on accessing the communication capabilities of the Mosaic (a few assembly instructions per message). The full performance potential of the machine is accessible to the user.
- *Expressivity:* The whole set of features that made C++ the object-oriented notation of choice for a growing number of complex applications is supported: inheritance, protection, overloading, as well as well-defined mechanisms to circumvent the default behavior. The full expressive power necessary for application programs is also accessible to the system programmer. A very clean virtual-machine model can be expressed: one or more instances of an entire runtime system can be run simply as another application.
- *Portability:* The bulk of the code is in the C+− to C++ translator, which is completely machine-independent. To support a particular machine or operating system, all that is required is a small library of C+− functions that define access to the communication capabilities.

2.3 MADRE: The Mosaic Runtime System

Nan Boden, Chuck Seitz

Organizational Philosophy. The goal of the Mosaic runtime system is to provide efficient and robust *distributed* runtime support for user programs. A fine-grain runtime system is composed of *component* runtime systems, one per node, that communicate and cooperate to manage their collective resources. The component itself can be implemented as a collection of fine-grain processes. Since the physical placement of these processes is not constrained within the programming model, we can place these runtime system processes so that each

node contains a set of *kernel* processes. Additional processes that help manage the resources of an individual node, called *remote* processes, may reside on other physical nodes.

By dividing each component runtime system into processes that are spread across the nodes of a multicomputer, we implement a fine-grain runtime system as a fine-grain program. Rather than the runtime system being a lump of code that exists to bridge a gap between machine and programming system, the fine-grain runtime system is itself a fine-grain program running on a fine-grain machine. Conceptually, a user computation is a collection of processes that is managed by runtime system processes. Since a process is an abstraction of a multicomputer node, the hardware level can also be modeled as a collection of processes. Thus, every level of computation can be expressed as a fine-grain program.

MADRE. The MADRE (**M**os**A**ic **D**istributed **R**untim**E**) system is a prototype runtime system developed using this design philosophy. Expressing the capability of each component runtime system using processes permits a highly modular approach to configuring a runtime system. Essential capabilities for message passing and process creation can be augmented with other capabilities, such as termination detection and special message-passing protocols, by instantiating the appropriate runtime system processes. Mechanisms for enhancing the robustness of the system, *eg*, handling message-receive-queue overflow or distributed code management, can also be included by adding processes. The ability to construct different runtime system configurations reliably and easily is critical to the Mosaic project due to the large spectrum of practical machine configurations and applications.

Implementation. MADRE is written in C+−, the programming notation described in the previous section. With its abstractions for processes and messages, C+− is a ideal notation for expressing the MADRE program. Since the programming and runtime-system levels are distinct, this choice of notation for the runtime system does not necessarily affect the choice of programming notation for user programs. Any programming notation whose base model is processes and messages can be supported directly by the runtime system. Currently, however, the MADRE system runs atop Mosaic C nodes and supports user programs written in C+−.

Experimental Evaluation. C+−, MADRE, and the Mosaic C multicomputer form the experimental apparatus used to investigate our ideas about fine-grain runtime-system design and organization, and our ideas for automatic resource management. We are currently developing a suite of benchmark programs constructed deliberately to stress the runtime system algorithms in the areas of process placement and robust resource management. By measuring the performance of the runtime system in executing these programs, we can evaluate the various runtime-system algorithm alternatives that have been developed.

3. Concurrent Computation

3.1 Affinity

Craig S. Steele, Chuck Seitz

Affinity is the new name given to the previously-described Page Kernel concurrent-programming system. It is now documented in a thesis and technical report, Caltech-CS-TR-92-08, briefly summarized below.

The Affinity Computational Model. Affinity is an experiment to explore a simple, convenient, and expressive programming model that provides adequate power for complex programming tasks while setting few constraints on potential concurrency. Although the programmer is required to formulate a computational problem explicitly in terms of medium-sized pieces of data and code, most of the additional functions necessary for concurrent execution are implicit. The execution of the light-weight, reactive processes, called *actions*, implicitly induces atomicity and consistency of data modifications. The programmer accesses shared data structures in a shared-memory fashion, but without the need for explicit locking to manage the problems of concurrent access and mutual exclusion. Program control flow is distributed and implicit.

The new name given to the programming model, Affinity, has a definition, "causal connection or relationship," that is fitting to the way programs are structured and scheduled.

Principal Features of the Model. Affinity consistency and coherence properties provide a tractable discipline for the dangerous power of a concurrent, shared-memory, programming style. Existing programming complexity-management techniques, such as object-oriented languages, can be used in this multicomputer environment. Affinity programs can compute consistent and correct results despite staleness of data, and asynchrony and nondeterminism, in execution of code. Program correctness is invariant under replication, or *cloning*, of actions. This aspect of the model yields a simple and robust mechanism for fault tolerance.

Experimental Implementation. The practicality of the Affinity programming model has been demonstrated by an implementation on a second-generation medium-grain multicomputer, the Ametek S/2010. The implementation is distributed, scalable, insensitive to network latency, and reasonably efficient. Action cloning has been shown to be an effective mechanism for continuing a computation through a single-node failure.

3.2 Molecular Dynamics on the Mosaic

K. Esselink, Jan L. A. van de Snepscheut

We have investigated the feasibility of implementing a Molecular Dynamics code on the Mosaic. A program originally developed by Shell Research, Amsterdam, to run on a torus network of Transputers was ported to the Mosaic with minor modifications. The current implementation features simulation of different Lennard-Jones fluids in one universe. Each Mosaic node can contain up to 150 particles, giving an upper bound for the 16K-node machine of 2.5 million particles in one simulation. At the time of development, only a 56-node Mosaic was available, and simulations were done on 8000 particles. In the last case, one time step in the simulation takes 17 seconds, and this time is needed almost exclusively (98%) for the computation of the forces. This may come as no surprise, considering that all floating-point operations are done by software and the communication in the Mosaic is

particularly fast. Yet, less than six Mosaic nodes were necessary for the same performance as one floating-point T800 Transputer.

Graphs of the energy as a function of time can be obtained. Furthermore, we have added some basic timing routines, as well as graphical output. A special program makes the initial universes, placing the particles on an fcc-lattice (face-centered cube).

In the first implementation, communication occurred only between neighboring nodes. Adaptations were made to study the effect of defining a torus communications network with random assignments of the elements to the physical network. On a 56-node Mosaic, no effect of the expected contention could be observed. It will be interesting to study the effect on the large Mosaic.

3.3 Communication Primitives

K. Esselink, H. Peter Hofstee, Rustan Leino, Jan L. A. van de Snepscheut

The Mosaic provides point-to-point routing between any two nodes. For the special case where the program uses a fixed communication graph of low degree between nodes, dynamic allocation of buffer space to incoming messages can be avoided. The 'send' and 'receive' statements in the Mosaic's Pascal programming system did not allow this to be implemented as efficiently as desired. The 'receive' statement was split into two parts, a 'set_receive' that sets the required parameters, and a 'wait_receive' that suspends the process until a message has been received. A little protocol that enables the sender between the 'set_receive' and 'wait_receive' actions has been implemented to take advantage of the fixed communication graph.

3.4 Program Transformation

H. Peter Hofstee, Jan L. A. van de Snepscheut

The emphasis of this work is on the development of transformations that lead to efficient programs, and on the correctness of the transformations.

A number of concurrent programs have been developed through systematic program transformations. First, a sequential solution is obtained. Second, the variables are partitioned and the sequential solution is rewritten to minimize the use of statements that refer to variables in different partitions. Third, the statements are distributed over a number of processes that have shared variables. Fourth, the statements that refer to variables in different partitions are rewritten to use communication actions instead of shared variables. Fifth, termination detection is added, if required.

In passing, we note that an existing algorithm for load balancing was generalized easily by these techniques to work on a larger class of graphs (viz. DAGs instead of trees).

4. VLSI Design

4.1 Multicomputer Routing Networks

Michael J. Pertel, Chuck Seitz

Previous studies by other authors have claimed a performance advantage in using adaptive routing rather than dimension-order routing in multicomputer routing networks. Based upon these previous results, a VLSI implementation of adaptive routing was undertaken. A router architecture was devised, and simulators were developed to reproduce, refine, and extend the previous studies. This research led to many new discoveries and insights that are not confined merely to the choice of routing algorithm.

Network Simulation. A compact, efficient, and versatile network simulator has been developed and published in a technical report (Caltech-CS-TR-92-04). The simulator is written in less than 200 lines of C code, and a complete listing of the program is provided. Other research groups have reported using several-thousand-line programs for routing-network simulation. (For example, one of Bill Dally's papers on adaptive routing mentions that their simulator was 9000 lines of C code.) Such immense programs cannot easily be scrutinized for latent assumptions and errors. A short program can be presented along with its results, to allow both scrutiny of the code and reproduction of the results. The complexity of previous simulators did not correctly reflect the complexity of the problem. The compact simulator is exact, not approximate, and its brevity results from the discovery of previously unknown routing-network properties, which are described in the report.

The nearly three-orders-of-magnitude reduction in program complexity leads to a comparable increase in program efficiency. Other authors have published simulations of networks with hundreds of nodes, with the assumption that the performance of such small networks could be extrapolated to networks with tens of thousands of nodes. The new simulator is fast enough to simulate large networks — such as the Mosaic's 128×128 mesh — on a SPARCstation.

The ability to simulate large networks has debunked previous misconceptions about routing-network behavior. These misconceptions arose from extrapolating the performance of small networks. The simulator is also more versatile than past simulators. Other authors have used separate simulators for adaptive and dimension-order routing, and unwittingly published results showing a performance advantage for adaptive routing that was actually an artifact of giving the adaptive routers more buffering than the dimension-order routers. Except for a single function that is only a few lines of code, the compact simulator uses the same program for dimension-order and adaptive routing.

Adaptive Routing. The notion that minimal-path forms of adaptive routing outperform dimension-order routing is refuted in a recent report (Caltech-CS-TR-92-06). Other authors have reported that adaptive routing supports higher throughput than dimension-order routing, but this is incorrect. Their results were an artifact of making an unequal comparison between adaptive and dimension-order routing; more than the routing algorithm was varied. With all other factors equal, one cannot do better than dimension-order routing; its throughput is optimal. The throughput of every form of minimal-path adaptive routing on meshes that we have studied is actually lower than that of dimension-order routing, and the complexity of these adaptive routers is substantially greater.

Claims that adaptive routing reduces latency are also wrong. For practical applied loads, dimension-order routing provides lower latency than adaptive routing. When the applied load to the routing network is a *very* small fraction of the network's capacity, adaptive

routing affords a minute decrease in latency, but network performance is irrelevant in this regime. Network performance is important for communication-limited computations, when nearly the full network capacity is used. The applied load at which latency grows without bound is lower for adaptive than for dimension-order routing; thus, dimension-order routing provides lower latency for heavy traffic.

Claims that adaptive routing provides fault tolerance are specious, and such claims are critiqued in the report. Because adaptive routing allows a packet to follow any minimal path through the network, it may actually make it harder to guarantee packet delivery in a faulty network. Adaptive routing does not improve network reliability, and, in some cases, can actually reduce reliability.

Choice of Dimension. The first multicomputer routing networks used the hypercube topology. It was later realized that the hypercube was not the best topology. Low-dimensional networks provide significantly better performance for the same or lower cost. The superiority of low-dimensional networks was clear to multicomputer designers from a variety of engineering perspectives, but a concise answer to the question "What is the optimal dimension for a multicomputer routing network?" was lacking.

A analysis technique was proposed to answer this question, and to convince hypercube enthusiasts that low-dimensional networks yield superior performance. The technique was based upon calculating the network dimension that minimized average latency for a given number of nodes and cost, where cost was estimated by wire bisection. This approach provided a compact argument for abandoning hypercubes, but it was not an adequate answer to the optimal-dimension question. The latency-minimization argument was based upon an incorrect understanding of network behavior, and many of its assumptions are now known to be false.

As evidenced by the ubiquity of pipelines, throughput is generally regarded as the primary performance metric, whereas latency is of secondary importance. Latency-optimization was premised upon the misconception that networks of equal cost would have equal throughput, so the "optimal" network would be determined by secondary characteristics. In reality, for a fixed network cost, there *is* a network dimension that maximizes throughput. *That* dimension should be regarded as the optimum.

The latency-optimization technique also had technical flaws. The technique was based upon a formula for latency in an uncongested network, but that formula is both quantitatively and qualitatively wrong for real networks. Unless the network is over-designed by a factor of nearly one-hundred, congestion not only cannot be neglected, but actually dominates network latency. The latency minimization was mathematically ill-posed because no latency-optimal dimension can be computed without specifying the network load, and the latency-optimal dimension for fixed wire bisection approaches 1 as the load approaches 100%.

When comparing networks of differing dimension, it is important to compare networks of equal cost. The latency-optimization technique used wire bisection as the network-cost measure; however, wire bisection is not a sufficiently realistic measure of network cost. The generally accepted cost metric for planar wiring is layout area. A report now in preparation (Caltech-CS-TR-92-09) provides an answer to the optimal-dimension question; it shows that two-dimensional networks are throughput-optimal for fixed area; similarly, three-dimensional networks are throughput-optimal for fixed volume.

4.2 The Routing-Chip Project

Chuck Seitz, Jakov Seizovic, Wen-King Su

The functions provided by the Mosaic router and packet interface are potentially useful to companies and research projects that do not have the time or capability to develop their own variants of these designs. Indeed, in addition to licensing the routing-chip technology to two companies, we have been providing routing chips to many other projects, as designated by DARPA, but only in one configuration. This "Caltech mesh-routing chip" employs 8-bit-parallel flow-control units (flits), and requires a fast interface between the self-timed channels to and from the node. The internal design of the new "Elko" router (see section 2.1.1) is substantially cleaner and more modular than that of earlier ("Ginzu" and "Frontier") routers; it is, accordingly, much easier to generate parametric variants of this design. Variants of the Mosaic packet interface would be able to provide a wider, hence less demanding, synchronous interface to and from the node.

Accordingly, with strong encouragement from our DARPA program managers, we have set out to develop a *family* of Elko routing, communication, and interface chips that can be used by other projects and licensed for use in commercial products. This family of chips includes:

- *EMRC2-2D8:* – is a 2D router with 8-bit-wide self-timed NEWS channels, and the same 8-bit-wide self-timed channels to and from the node. This router is a new design, but is backward- and pin-compatible with the Ginzu and Frontier routers provided to numerous other projects. The specifications for this router are included as an attachment to this report.
- *EMRC2-2D8B:* – is a variant of the EMRC2-2D8 that supports packet broadcast.
- *EMRC2-2D9:* – is a variant of the EMRC2-2D8 with 9-bit-wide channels. The extra bit may be used for parity or other purposes.
- *EMRC2-1D16:* – is a 1D, 16-bit-wide variant of the EMRC2-2D8. The pinout for packaging this chip in a 132PGA is the same as that of the Frontier router in the Stanford DASH multiprocessor.
- *EMRC-SP:* is a variant of the EMRC2-2D8 with synchronous, full-duplex channels to and from the node, and timeouts on the NEWS and node channels. This chip has been licensed to a company that plans to use it in a commercial product.
- *Slack20:* – converts between the 0-slack (non-interference) protocol generally used on the NEWS channels and a 20-slack protocol that allows full bandwidth to be maintained when driving long cables.
- *ERI-36:* – is an interface between 8- or 9-bit routers and a 32- or 36-bit synchronous bus. This chip includes 9$\leftrightarrow$36 conversion, pipeline synchronizers, and synchronous FIFOs between the router channels and the bus.
- *ERI-72:* – is similar to the ERI-36, but is an interface between 8- or 9-bit routers and a 64- or 72-bit synchronous bus.
- *EMRC2-2D9P36:* – is an EMRC2-2D9 and ERI-36 combined on a single chip.
- *RRI-16P32:* – is a ring router with two virtual channels per 16-bit channel on the ring, and a 32-bit synchronous bus interface to the node.

Detail specifications are available by email if you send your router requirements to chuck@vlsi.cs.caltech.edu. Additional routing and communication chips may be added to this family as required, and as we get new ideas.

The layouts, simulations, and Verilog models for these chips are being developed this summer by a team of four of the best students out of this year's VLSI-design class. In addition to the technology-transfer motivations for this project, we are using these designs as vehicles for experimenting with several methods of extending the functions and improving the performance of these chips.

4.3 Asynchronous GaAs

José A. Tierno, Alain J. Martin

Within the past year, we were able to design and fabricate a 16-bit microprocessor running at 70 MIPS, and a small static RAM with an access time of 3ns. We have also developed a new logic family, Source-Follower-FET-Logic, which is more robust and more versatile, but also more complex, than the standard DCFL. We expect a mixed SFFL/DCFL approach to give excellent results.

With an electron mobility about six times that of silicon at room temperature and for low electric field, and with a lower parasitic capacitance due to a semi-insulating substrate, GaAs is potentially significantly faster than silicon. Up until recently however, GaAs was not available to the VLSI community at large because of inherent fabrication difficulties. These difficulties seem to have been overcome to a large extent. Several foundries are now offering GaAs fabrication lines under conditions similar to CMOS, with density limited to less than 100,000 transistors. In particular, Vitesse Semiconductors is offering fabrication through MOSIS to the DARPA community.

At the moment, the transistor of choice for GaAs digital VLSI is the MESFET. Because there is no oxide isolating the gate of a transistor from source and drain, the different logic families available are much less attractive than CMOS or even nMOS. Because of the small voltage swing, the noise margin is limited, and the fanin and fanout of the gates are restricted. With no complementary logic available, the logic is ratioed. As a result, a considerable fraction of the speed advantage is lost because of the complexity of the available logic families compared to CMOS.

So far, the challenge of realizing high-speed digital VLSI in GaAs has attracted only a few groups in the research community. Probably, the majority of the researchers make the safe bet that the expected, continuing improvements in density and die size for CMOS will close the gap between CMOS and GaAs. But there is undoubtedly a window of opportunity now for high-speed VLSI in GaAs.

Our interest in GaAs is motivated slightly differently. We have developed a design method for asynchronous VLSI that is, to a large extent, independent of the technology; thus, it should be straightforward to port a design from one technology to another. Also, since the circuits designed are delay-insensitive, they are more robust to variations of the physical parameters. Hence, the method should make it easier to design in a demanding technology such as GaAs, where parameters – particularly threshold voltages – are difficult to control. Finally, since the circuits we design don't use a clock, we avoid the difficult problem of high-speed clocking schemes. Hence, adapting our method to GaAs design would be an excellent demonstration of the advantages of the method.

From the onset, our intention was to port to GaAs the asynchronous microprocessor we designed in CMOS in 1989. We would kill two birds with one stone as we would demonstrate

the portability of our approach across vastly different technologies, and the efficiency and robustness of our design method. At the same time, we would climb the learning curve rapidly by tackling a difficult example as a first design. And, last but not least, we would have a microprocessor in the 100 MIPS range.

4.3.1 A New Logic Family

The logic family most commonly used with MESFET GaAs, and promoted by Vitesse, is called DCFL. DCFL is similar to nMOS, and is thus very simple. But, unfortunately, because of the small voltage range available, it is limited in terms of noise margins, fanin, fanout, and the number of gates that can practically be built.

Good noise immunity is important for asynchronous design, since the transitions from one voltage level to the other have to be monotonic. Also, the logic must allow for the direct construction of a family of gates called *generalized C-elements* that are the basic building blocks of the control part in asynchronous logic. Constructing generalized C-elements from NOR gates is problematic and inefficient; hence, we came to the conclusion that the existing logic families, DCFL in particular, were not suited for asynchronous design, and that we should invent our own.

The design of this logic family is the work of José Tierno, and is described in detail in his MS thesis (Caltech-CS-TR-92-19). It is a so-called *source-follower* logic, and is therefore named SFFL for source-follower-FET-logic. It offers the generality and robustness that we were looking for: We can implement any pair of non-interfering production rules, *ie*, any pair of pull-up and pull-down conditions for a node that are not short-circuiting. The price we pay is that the logic is more complex than DCFL, and some of the ratio requirements on transistor sizes are two-sided.

Although we can generate any gate, we have constructed a library of standard gates that we have used in several designs already. This library is not exhaustive; however, only four extra gates of a total of 170 needed to be added for the microprocessor control.

4.3.2 The GaAs Asynchronous Microprocessor

We implemented the Caltech Asynchronous Microprocessor in GaAs using SFFL. The design was started in early 1991, a first version was submitted for fabrication in July 1991, and a second, corrected, version (see figure 6) was submitted for fabrication in December 1991.

The first version was not functional because of a rather trivial but fatal layout bug in the output pad drivers. This bug made it impossible to get any signals out of the chip. This pad-driver problem and a few other minor problems were corrected for the December run.

Out of the 29 bonded chips that we received from this run, only two were found to be fully functional at room temperature. The problem was traced to a specific temperature-dependent circuit in the implementation of the buses. The subthreshold current increases with temperature, and pull-down transistors that should have been cut off were pulling down the bus. This hypothesis was first confirmed by extensive SPICE simulations. Subsequent testing with the chips cooled with a freon spray showed that 13 chips are fully functional.

The working chips have a cycle time of 14ns. The power dissipation is 4.5W at 2.3V. With a speed below 70 MIPS, we didn't reach the expected factor 5 increase in speed from the CMOS version, which would have put us into the 100 MIPS range. But, we just wanted to have a functional chip, which seems to be the case.

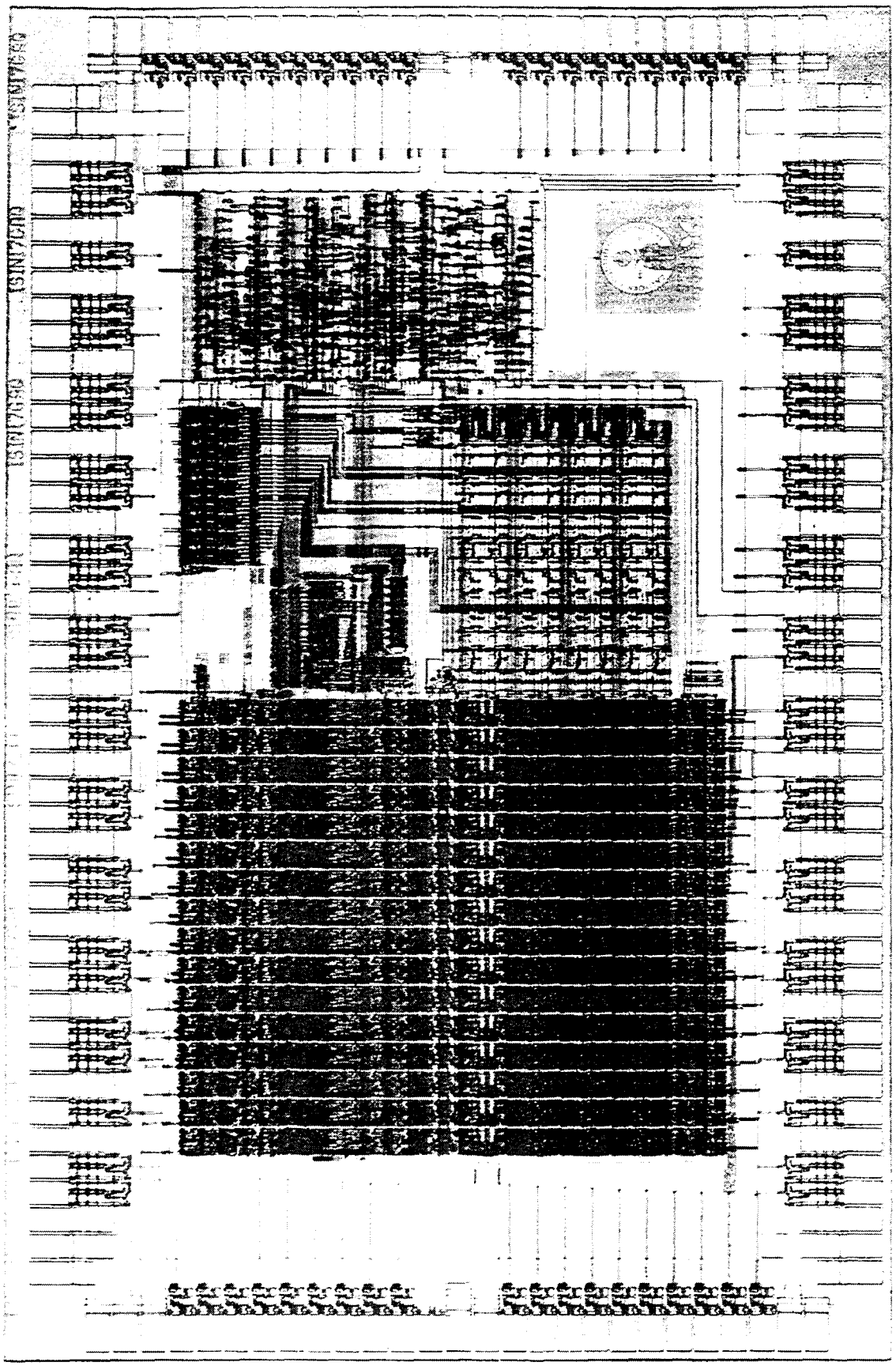


Figure 6: Photomicrograph of the GaAs Asynchronous Microprocessor.

4.3.3 Asynchronous Static Memories

With the speed of computer systems being more and more memory-limited, the design of fast, static memories should attract the attention of the VLSI community. We believe that an asynchronous approach to the design of static RAMs should have important speed advantages in spite of the inherent area overhead of asynchronous design. In a purely delay-insensitive design style, the writing of a single bit of data has to be acknowledged by a circuitry that requires at least 7 transistors, but less conservative designs are, of course, possible.

We designed and fabricated two different types of static memories in GaAs. The first is a dual-ported register file, 16 words of 4 bits/word. It was meant to provide a small amount of very fast memory for the asynchronous GaAs processor to run programs at high speed. Out of 30 chips received, 29 were found fully functional. Access time is 5ns, and the chip dissipates 500mW at 2.2V.

The second static RAM has 64 words of 4 bits/word, and was designed as a stepping stone towards a larger memory to be used as a cache for the asynchronous GaAs processor. All 30 chips received were found to be functional. The access time is 3ns, and the chip dissipates 700mW at 2.3V.

This 64×4 memory was designed after the processor, and incorporates several improvements derived from our experience with the processor design. It uses mixed logic: SFFL for the control and DCFL for the core of the memory. Also, the circuits were carefully optimized for high speed and low power consumption. The performance obtained indicates that the improvements envisioned for the next version of the microprocessor should give good results.

4.3.4 Preliminary Conclusions

We can draw a number of preliminary conclusions from this first phase of the project. Because of the reduced turn-around time for MOSIS GaAs runs, we decided to skip test chips, and went right for a conservative version of the microprocessor. Within a year, we had a working 16-bit microprocessor and a small static RAM. This experience indicates that GaAs is indeed usable for (medium-size) digital VLSI. It also shows the flexibility of our synthesis method, since we were able to adapt all of our CAD tools to GaAs, and port the designs of two complex CMOS chips (microprocessor and RAM) to GaAs successfully.

The logic family we have designed (SFFL) is now stable and well understood. We can generate directly any circuit described by production rules (programs). A fairly complete library of standard cells is available. We have a good idea of how these circuits behave under a wide variation of parameters, and we know how to optimize them for high speed and low power consumption.

We are not concerned by the fact that we got "only" 70 MIPS for this first asynchronous GaAs processor. We know that we have made a number of optimization errors. We also know that an entirely SFFL approach is overly conservative. The experiment with the static RAM indicates that a mixed DCFL/SFFL logic should give excellent results. We expect to double the performance of the processor on the next design.

4.4 Testing Asynchronous Circuits

Pieter Hazewindus, Alain Martin

One of the arguments most often used against asynchronous techniques is testing. There is an old belief in the VLSI-design community that testing asynchronous circuits is very difficult

because of races and hazards. Strangely enough, the opposite belief has been alive in the self-timed-design community; namely, that asynchronous circuits are easy to test because a stuck-at fault results in the circuit halting. We have studied the problem of asynchronous circuit testing thoroughly, and have concluded that testing of asynchronous circuits is neither more difficult nor simpler than testing clocked circuits. The techniques are actually quite similar. What we have also discovered is that the testing of circuits synthesized from a high-level description (program) is greatly simplified; the test vectors can be generated directly from the high-level description.

The method we have developed to test delay-insensitive circuits uses the single stuck-at fault model. These circuits are synthesized from a high-level specification. Since the circuits are hazard-free by construction, there is no need to test for hazards in the circuit. Most faults cause the circuit to halt during test, since they cause an acknowledgment not to occur when it should. But there are stuck-at faults that do not cause the circuit to halt under any condition. These are *stimulating* faults; they cause a premature firing of a production rule. For such a stimulating fault to be testable, the premature firing has to be propagated to a primary output. If this is not guaranteed to occur, then one or more test points have to be added to the circuit. Any stuck-at fault is testable, with the possible addition of test points. For combinational delay-insensitive circuits, finding test vectors is reduced to the same problem as exists for synchronous combinational logic. For sequential circuits, the synthesis method is used to find a test for each fault efficiently, to find the location of the test points, and to find a test that detects all faults in a circuit.

The number of test points needed to fully test the circuit is very low, and the size of the additional testing circuitry is small. A test derived with a simple transformation of the handshaking expansion yields high fault coverage. Adding tests for the remaining faults results in a small complete test for the circuit.

4.5 Asynchronous Adders

Tony Lee, Alain Martin

Integer addition is the most important of all arithmetic operations. Not only is it needed to perform more complicated operations, such as multiplication and floating-point arithmetic functions, but it is also used for incrementing program counters and calculating memory addresses. Thus, much effort has been spent on designing fast and efficient adders. For traditional clocked designs, the actual performance of an adder is not critical as long as its worst-case latency fits within the allowed clock period. However, for asynchronous designs, the average latency usually has a larger bearing on the overall performance of the system.

We have been studying different implementations of delay-insensitive adders, and have developed production-rule sets for several designs in each of the four traditional classes: ripple-carry adders, carry-lookahead adders, carry-skip adders, and carry-select adders.

Our goal is to be able to evaluate each design from its production rule set without resorting to explicit layouts. We have already done some simple measurements on each design, such as counting the number of elements and transistors, estimating the wiring area needed to transmit signals across bits, and finding the number of signal transitions on the critical path for both the worst case and average case. We are now looking for a more refined timing model to better estimate the performance of each design.

California Institute of Technology
Computer Science Department, 256-80
Pasadena CA 91125

Technical Reports

1 July 1992

Prices include postage and help to defray our printing and mailing costs.

Publication Order Form

To order reports fill out the last page of this publication form. *Prepayment* is required for all materials. Purchase orders will not be accepted. All foreign orders must be paid by international money order or by check for a minimum of $50.00 drawn on a U.S. bank in U.S. currency, payable to CALTECH.

CS-TR-92-16	$8.00	Constructing some Distributed Programs Hofstee, H. Peter
CS-TR-92-15	$22.00	VLSI Analogs of Neuronal Visual Processing: A Synthesis of Form and Function, (PhD Thesis) Mahowald, Misha
CS-TR-92-14	$15.00	Testing Delay-Insensitive Circuits, (PhD Thesis) Hazewindus, Pieter J
CS-TR-92-13	$5.00	Compositional C++: Compositional Parallel Programming Chandy, K Mani; Kesselman, Carl
CS-TR-92-11	$6.00	Mutual Exclusion on a Token Ring in C++: Program and Proof Binau, Ulla
CS-TR-92-07	$6.00	A Compiler Approach to Scalable Concurrent Program Design Foster, Ian; Taylor, Stephen
CS-TR-92-06	$6.00	A Critique of Adaptive Routing Pertel, Michael J
CS-TR-92-05	$5.00	Mesh Distance Formulae Pertel, Michael J
CS-TR-92-04	$6.00	A Simple Simulator for Multicomputer Routing Networks Pertel, Michael J
CS-TR-92-03	$5.00	A Delay-Insensitive Multiply-Accumulate Unit Nielsen, Christian D; Martin, Alain J
CS-TR-91-11	$12.00	An Object-Oriented Real-Time Simulation of Music Performance Using Interactive Control, (PhD Thesis) Dyer, Lounette M
CS-TR-91-10	$6.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-91-09	$8.00	Combinatorial Design of Fault-Tolerant Communication Structures, with Applications to Non-Blocking Switches, (PhD Thesis) Schweizer, David L
CS-TR-91-07	$4.00	Inversion of a Recursive Tree Traversal van de Snepscheut, Jan L A
CS-TR-91-06	$6.00	On the Correctness of Sliding Window Protocols van de Snepscheut, Jan L A
CS-TR-91-05	$6.00	A Distributed Implementation of a Task Pool Hofstee, H Peter; Lukkien, Johan J; van de Snepscheut, Jan L A
CS-TR-91-04	$4.00	A LISP Programming Exercise van de Snepscheut, Jan L A
CS-TR-91-03	$6.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-91-02	$6.00	A Tutorial Introduction to Mosaic Pascal Lukkien Johan J; van de Snepscheut, Jan L A

CS-TR-91-01	$12.00	Performance Analysis and Optimization of Asynchronous Circuits, (PhD Thesis) Burns, Steven M
CS-TR-90-18	*$4.00*	Performance Analysis and Optimization of Asynchronous Circuits Burns, Steven M; Martin, Alain J
CS-TR-90-17	$4.00	Testing Delay-Insensitive Circuits Martin, Alain J; Hazewindus Pieter J
CS-TR-90-16	$14.00	Parallel Program Design and Generalized Weakest Preconditions Lukkien, Johan J
CS-TR-90-15	$14.00	A Unified Framework for Constraint-based Modeling, (PhD Thesis) Kalra, Devendra
CS-TR-90-14	$6.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-90-13	$7.00	Weakest Preconditions for Progress Lukkien, Johan J; van de Snepscheut, Jan L A
CS-TR-90-12	$5.00	Performance Analysis and Optimization of Asynchronous Circuits Produced by Martin Synthesis Burns, Steven M
CS-TR-90-11	$4.00	Characterizing NP and Measuring Instance Complexity Judd, J Stephen
CS-TR-90-10	$10.00	Primer for Program Composition Notation Chandy, K Mani; Taylor, Stephen
CS-TR-90-09	$4.00	Asynchronous Circuits for Token-Ring Mutual Exclusion Martin, Alain J
CS-TR-90-07	$4.00	Compiler Optimization of Array Data Storage Gupta, Rajiv; Kajiya, James T
CS-TR-90-06	$4.00	Distributed Sorting Hofstee, H Peter; Martin, Alain J; van de Snepscheut, Jan L A
CS-TR-90-05	$6.00	Submicron Systems Architecture ARPA Semiannual Technical Report
CS-TR-90-03	$6.00	Program Composition Project Chandy, K Mani; Taylor, Stephen; Kesselman, Carl; Foster, Ian
CS-TR-90-02	$4.00	Limitations to Delay-Insensitivity in Asynchronous Circuits Martin, Alain J
CS-TR-90-01	$6.00	Properties of the V-C Dimension, (MS Thesis) Fyfe, Andrew
CS-TR-89-13	$6.00	Communication Behavior of Linear Arrays of Processes, (MS Thesis) Lee, Tak K
CS-TR-89-12	$6.00	Submicron Systems Architecture ARPA Semiannual Technical Report
CS-TR-89-11	$18.00	Reactive-Process Programming and Distributed Discrete-Event Simulation, (PhD Thesis) Su, Wen-King
CS-TR-89-10	$14.00	Silicon Models of Early Audition, (PhD Thesis) Lazarro, John
CS-TR-89-09	$30.00	Framework for Adaptive Routing in Multicomputer Networks, (PhD Thesis) Ngai, John Y
CS-TR-89-07	$12.00	Constraint Methods for Neural Networks and Computer Graphics, (PhD Thesis) Platt, John C
CS-TR-89-06	$2.00	First Asynchronous Microprocessor: The Test Results Martin, Alain J; Burns, Steven M; Lee, Tak K; Borkovic, Drazen; Hazewindus, Pieter J
CS-TR-89-05	$4.00	Essence of Distributed Snapshots Chandy, K Mani

CS-TR-89-04	$10.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-89-03	$6.00	Feature-oriented Image Enhancement with Shock Filters, I Rudin, Leonid I; Osher, Stanley
CS-TR-89-02	$6.00	Design of an Asynchronous Microprocessor Martin, Alain J
CS-TR-89-01	$8.00	Programming in VLSI From Communicating Processes to Delay-insensitive Circuits Martin, Alain J
CS-TR-88-22	$4.00	Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm Su, Wen-King; Seitz, Charles L
CS-TR-88-21	$6.00	Winner-Take-All Networks of O(N) Complexity Lazzaro, J; Ryckebusch, S; Mahowald, M A; Mead, C A
CS-TR-88-20	$14.00	Neural Network Design and the Complexity of Learning Judd, J Stephen
CS-TR-88-19	$10.00	Controlling Rigid Bodies with Dynamic Constraints Barzel, Ronen
CS-TR-88-18	$6.00	Submicron Systems Architecture Project ARPA Semiannual Technical Report
CS-TR-88-17	$6.00	Constrained Differential Optimization for Neural Networks Platt, John C; Barr, Alan H
CS-TR-88-16	$6.00	Programming Parallel Computers Chandy, K Mani
CS-TR-88-15	$26.00	Applications of Surface Networks to Sampling Problems in Computer Graphics, (PhD Thesis) Von Herzen, Brian
CS-TR-88-14	$8.00	Syntax-directed Translation of Concurrent Programs into Self-timed Circuits Burns, Steven M; Martin, Alain J
CS-TR-88-13	$4.00	Message-Passing Model for Highly Concurrent Computation Martin, Alain J
CS-TR-88-12	$8.00	Comparison of Strict and Non-strict Semantics for Lists, (MS Thesis) Burch, Jerry R
CS-TR-88-11	$10.00	Study of Fine-Grain Programming Using Cantor, (MS Thesis) Boden, Nanette J
CS-TR-88-10	$6.00	Reactive Kernel, (MS Thesis) Seizovic, Jakov
CS-TR-88-07	$6.00	Hexagonal Resistive Network and the Circular Approximation Feinstein, David I
CS-TR-88-06	$6.00	Theorems on Computations of Distributed Systems Chandy, K Mani
CS-TR-88-05	$6.00	Submicron Systems Architecture ARPA Semiannual Technical Report
CS-TR-88-04	$6.00	Cochlear Hydrodynamics Demystified Lyon, Richard F; Mead, Carver A
CS-TR-88-03	$8.00	PS: Polygon Streams: A Distributed Architecture for Incremental Computation Applied to Graphics. (MS Thesis) Gupta, Rajiv
CS-TR-88-02	$8.00	Automated Compilation of Concurrent Programs into Self-timed Circuits, (MS Thesis) Burns, Steven M
CS-TR-88-01	$6.00	C Programmer's Abbreviated Guide to Multicomputer Programming Seitz, Charles L; Seizovic, Jakov; Su, Wen-King

5258:TR:87	$6.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5256:TR:87	$4.00	Synthesis Method for Self-timed VLSI Circuits Martin, Alain J. *(current supply only: see Proc. ICCD'87: 1987 IEEE Int'l. Conf. on Computer Design 224-229, Oct'87)*
5253:TR:87	$4.00	Synthesis of Self-timed Circuits by Program Transformation Burns, Steven M; Martin, Alain J
5251:TR:87	$4.00	Conditional Knowledge as a Basis for Distributed Simulation Chandy, K Mani; Misra, Jay
5250:TR:87	$20.00	Images, Numerical Analysis of Singularities and Shock Filters, (PhD Thesis) Rudin, Leonid I
5249:TR:87	$12.00	Logic from Programming Language Semantics, (PhD Thesis) Choo, Young-il
5247:TR:87	$12.00	VLSI Concurrent Computation for Music Synthesis, (PhD Thesis) Wawrzynek, John
5246:TR:87	$6.00	Framework for Adaptive Routing Ngai, John Y; Seitz, Charles L
5244:TR:87	$6.00	Multicomputers Athas, William C; Seitz, Charles L
5243:TR:87	$10.00	Resource-Bounded Category and Measure in Exponential Complexity Classes, (PhD Thesis) Lutz, Jack H
5242:TR:87	$16.00	Fine Grain Concurrent Computations, (PhD Thesis) Athas, William C
5241:TR:87	$6.00	VLSI Mesh Routing Systems, (MS Thesis) Flaig, Charles M
5240:TR:87	$4.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5239:TR:87	$6.00	Trace Theory and Systolic Computations Rem, Martin
5238:TR:87	$14.00	Incorporating Time in the New World of Computing System, (MS Thesis) Poh, Hean Lee
5236:TR:86	$8.00	Approach to Concurrent Semantics Using Complete Traces, (MS Thesis) Van Horn, Kevin S
5235:TR:86	$8.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5234:TR:86	$6.00	High Performance Implementation of Prolog Newton, Michael O
5233:TR:86	$6.00	Some Results on Kolmogorov-Chaitin Complexity, (MS Thesis) Schweizer, David L
5232:TR:86	$8.00	Cantor User Report Athas, William C; Seitz, Charles L
5230:TR:86	$48.00	Monte Carlo Methods for 2-D Compaction, (PhD Thesis) Mosteller, Richard C
5229:TR:86	$8.00	anaLOG — A Functional Simulator for VLSI Neural Systems, (MS Thesis) Lazzaro, John
5228:TR:86	$6.00	On Performance of k-ary n-cube Interconection Networks Dally, William J
5227:TR:86	$36.00	Parallel Execution Model for Logic Programming, (PhD Thesis) Li, Pey-yun

5223:TR:86	$30.00	Integrated Optical Motion Detection, (PhD Thesis) Tanner, John E
5221:TR:86	$12.00	Sync Model: A Parallel Execution Method for Logic Programming Li, Pey-yun; Martin, Alain J. *(current supply only: see Proc SLP'86 3rd IEEE Symp on Logic Programming Sept '86)*
5220:TR:86	$8.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5215:TR:86	$4.00	How to Get a Large Natural Language System into a Personal Computer Thompson, Bozena H; Thompson, Frederick B
5214:TR:86	$4.00	ASK is Transportable in Half a Dozen Ways Thompson, Bozena H; Thompson, Frederick B
5212:TR:86	$4.00	On Seitz' Arbiter Martin, Alain J
5210:TR:86	5.00	Compiling Communicating Processes into Delay-Insensitive VLSI Circuits Martin, Alain J
5207:TR:86	$4.00	Complete and Infinite Traces: A Descriptive Model of Computing Agents Van Horn, Kevin S
5205:TR:85	$4.00	Two Theorems on Time Bounded Kolmogrov-Chaitin Complexity Schweizer, David L; Abu-Mostafa, Yaser
5204:TR:85	$6.00	An Inverse Limit Construction of a Domain of Infinite Lists Choo, Young-Il
5202:TR:85	$30.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5200:TR:85	$36.00	ANIMAC: A Multiprocessor Architecture for Real-Time Computer Animation, (PhD Thesis) Whelan, Daniel S
5198:TR:85	$16.00	Neural Networks, Pattern Recognition and Fingerprint Hallucination, (PhD Thesis) Mjolsness, Eric
5197:TR:85	$14.00	Sequential Threshold Circuits, (MS thesis) Platt, John C
5195:TR:85	$6.00	New Generalization of Dekker's Algorithm for Mutual Exclusion Martin, Alain J. *(current supply only: see Information Processing Letters 23 295-297 1986)*
5194:TR:85	$10.00	Sneptree — A Versatile Interconnection Network Li, Pey-yun; Martin, Alain J
5193:TR:85	$4.00	Delay-Insensitive Fair Arbiter Martin, Alain J
5190:TR:85	$6.00	Concurrency Algebra and Petri Nets Choo, Young-il
5189:TR:85	$20.00	Hierarchical Composition of VLSI Circuits, (PhD Thesis) Whitney, Telle E
5185:TR:85	$22.00	Combining Computation with Geometry, (PhD Thesis) Lien, Sheue-Ling
5184:TR:85	$14.00	Placement of Communicating Processes on Multiprocessor Networks, (MS Thesis) Steele, Craig
5179:TR:85	$6.00	Sampling Deformed, Intersecting Surfaces with Quadtrees, (MS Thesis) Von Herzen, Brian P
5178:TR:85	$18.00	Submicron Systems Architecture ARPA Semiannual Technical Report
5174:TR:85	$14.00	Balanced Cube: A Concurrent Data Structure Dally, William J; Seitz, Charles L

5172:TR:85	$12.00	**Combined Logical and Functional Programming Language** Newton, Michael O
5168:TR:84	$6.00	**Object Oriented Architecture** Dally, William J; Kajiya, James T
5165:TR:84	$8.00	**Customizing One's Own Interface Using English as Primary Language** Thompson, Bozena H; Thompson, Frederick B
5164:TR:84	$26.00	**ASK French - A French Natural Language Syntax**, (MS Thesis) Sanouillet, Remy
5160:TR:84	$14.00	**Submicron Systems Architecture** ARPA Semiannual Technical Report
5158:TR:84	$12.00	**VLSI Architecture for Sound Synthesis** Wawrzynek, John; Mead, Carver A
5157:TR:84	$30.00	**Bit-Serial Reed-Solomon Decoders in VLSI**, (PhD Thesis) Whiting, Douglas
5147:TR:84	$8.00	**Networks of Machines for Distributed Recursive Computations** Martin, Alain J; van de Snepscheut Jan L A
5143:TR:84	$10.00	**General Interconnect Problem**, (MS Thesis) Ngai, John Y
5140:TR:84	$10.00	**Hierarchy of Graph Isomorphism Testing**, (MS Thesis) Chen, Wen-Chi
5139:TR:84	$8.00	**HEX: A Hierarchical Circuit Extractor**, (MS Thesis) Oyang, Yen-Jen
5137:TR:84	$14.00	**Dialogue Designing Dialogue System**, (PhD Thesis) Ho, Tai-Ping
5136:TR:84	$10.00	**Heterogeneous Data Base Access**, (PhD Thesis) Papachristidis, Alex
5135:TR:84	$14.00	**Toward Concurrent Arithmetic**, (MS Thesis) Chiang, Chao-Lin
5134:TR:84	$4.00	**Using Logic Programming for Compiling APL**, (MS Thesis) Derby, Howard V
5133:TR:84	$26.00	**Hierarchical Timing Simulation Model for Digital Integrated Circuits and Systems**, (PhD Thesis) Lin, Tzu-mu
5132:TR:84	$20.00	**Switch Level Fault Simulation of MOS Digital Circuits**, (MS Thesis) Schuster, Michael D
5129:TR:84	$10.00	**Design of the MOSAIC Processor**, (MS Thesis) Lutz, Christopher P
5128:TM:84	$6.00	**Linguistic Analysis of Natural Language Communication with Computers** Thompson, Bozena H
5125:TR:84	$12.00	**Supermesh**, (MS Thesis) Su, Wen-King
5123:TR:84	$28.00	**Mossim Simulation Engine Architecture and Design** Dally, William J
5122:TR:84	$16.00	**Submicron Systems Architecture** ARPA Semiannual Technical Report
5114:TM:84	$6.00	**ASK As Window to the World** Thompson, Bozena H; Thompson, Frederick B
5112:TR:83	$44.00	**Parallel Machines for Computer Graphics**, (PhD Thesis) Ulner, Michael
5106:TM:83	$2.00	**Ray Tracing Parametric Patches** Kajiya, James T

5104:TR:83	$18.00	Graph Model and the Embedding of MOS Circuits, (MS Thesis) Ng, Tak-Kwong
5094:TR:83	$4.00	Stochastic Estimation of Channel Routing Track Demand Ngai, John Y
5092:TM:83	$4.00	Residue Arithmetic and VLSI Chiang, Chao-Lin; Johnsson, Lennart S
5091:TR:83	$4.00	Race Detection in MOS Circuits by Ternary Simulation Bryant, Randal E
5090:TR:83	$18.00	Space-Time Algorithms: Semantics and Methodology, (PhD Thesis) Chen, Marina C
5089:TR:83	$20.00	Signal Delay in General RC Networks with Application to Timing Simulation of Digital Integrated Circuits Lin, Tzu-Mu; Mead, Carver A
5086:TR:83	$8.00	VLSI Combinator Reduction Engine, (MS Thesis) Athas, William C
5082:TR:83	$20.00	Hardware Support for Advanced Data Management Systems, (PhD Thesis) Neches, Philip
5081:TR:83	$8.00	RTsim - A Register Transfer Simulator, (MS Thesis) Lam, James K
5074:TR:83	$20.00	Robust Sentence Analysis and Habitability Trawick, David
5073:TR:83	$24.00	Automated Performance Optimization of Custom Integrated Circuits, (PhD Thesis) Trimberger, Steven
5065:TR:82	$6.00	Switch Level Model and Simulator for MOS Digital Systems Bryant, Randal E
5054:TM:82	$6.00	Introducing ASK, A Simple Knowledgeable System Conf on App'l Natural Language Processing Thompson, Bozena H. and Frederick B Thompson
5051:TM:82	$4.00	Knowledgeable Contexts for User Interaction Proc Nat'l Computer Conference Thompson, Bozena H; Thompson, Frederick B; Ho, Tai-Ping
5035:TR:82	$18.00	Type Inference in a Declarationless, Object-Oriented Language, (MS Thesis) Holstege, John E
5034:TR:82	$24.00	Hybrid Processing, (PhD Thesis) Carroll, Christopher
5033:TR:82	$8.00	MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual Schuster, Michael D; Bryant, Randal E; Whiting, Douglas L
5029:TM:82	$8.00	POOH User's Manual Whitney, Telle E
5018:TM:82	$4.00	Filtering High Quality Text for Display on Raster Scan Devices Kajiya, James T; Ullner, Michael
5017:TM:82	$4.00	Ray Tracing Parametric Patches Kajiya, James T
5015:TR:82	$30.00	VLSI Computational Structures Applied to Fingerprint Image Analysis Megdal, Barry
5014:TR:82	$30.00	Extension of Object-Oriented Languages to a Homogeneous, Concurrent Architecture, (PhD Thesis) Lang, Charles R
5012:TM:82	$4.00	Switch-Level Modeling of MOS Digital Circuits Bryant, Randal E
5000:TR:82	$12.00	Self-Timed Chip Set for Multiprocessor Communication, (MS Thesis) Whiting, Douglas L

4684:TR:82	$6.00	**Characterization of Deadlock Free Resource Contentions** Chen, Marina C; Rem, Martin; Graham, Ronald
4655:TR:81	$40.00	**Proc Second Caltech Conf on VLSI** Seitz, Charles L
2276:TM:78	$24.00	**Language Processor and a Sample Language** Ayres, Ronald

Caltech Computer Science Technical Reports

Please PRINT your name, address and amount enclosed below:

name _____

Address _____

City _____ State _____ Zip _____ Country _____

Amount enclosed $ _____

_____ Please check here if you wish to be included on our mailing list

_____ Please check here for any change of address

_____ Please check here if you would prefer to have future publications lists sent to your e-mail address.

E-mail address _____

Return this form to: Computer Science Library, 256-80, Caltech, Pasadena CA 91125

___CS-TR-92-16	___CS-TR-90-10	___CS-TR-88-14	___5235:TR:86	___5184:TR:85	___5106:TM:83
___CS-TR-92-15	___CS-TR-90-09	___CS-TR-88-13	___5234:TR:86	___5179:TR:85	___5104:TR:83
___CS-TR-92-14	___CS-TR-90-07	___CS-TR-88-12	___5233:TR:86	___5178:TR:85	___5094:TR:83
___CS-TR-92-13	___CS-TR-90-06	___CS-TR-88-11	___5232:TR:86	___5174:TR:85	___5092:TM:83
___CS-TR-92-11	___CS-TR-90-05	___CS-TR-88-10	___5230:TR:86	___5172:TR:85	___5091:TR:83
___CS-TR-92-07	___CS-TR-90-03	___CS-TR-88-07	___5229:TR:86	___5168:TR:84	___5090:TR:83
___CS-TR-92-06	___CS-TR-90-02	___CS-TR-88-06	___5228:TR:86	___5165:TR:84	___5089:TR:83
___CS-TR-92-05	___CS-TR-90-01	___CS-TR-88-05	___5227:TR:86	___5164:TR:84	___5086:TR:83
___CS-TR-92-04	___CS-TR-89-13	___CS-TR-88-04	___5223:TR:86	___5160:TR:84	___5082:TR:83
___CS-TR-92-03	___CS-TR-89-12	___CS-TR-88-03	___5221:TR:86	___5158:TR:84	___5081:TR:83
___CS-TR-91-11	___CS-TR-89-11	___CS-TR-88-02	___5220:TR:86	___5157:TR:84	___5074:TR:83
___CS-TR-91-10	___CS-TR-89-10	___CS-TR-88-01	___5215:TR:86	___5147:TR:84	___5073:TR:83
___CS-TR-91-09	___CS-TR-89-09	___5258:TR:87	___5214:TR:86	___5143:TR:84	___5065:TR:82
___CS-TR-91-07	___CS-TR-89-07	___5256:TR:87	___5212:TR:86	___5140:TR:84	___5054:TM:82
___CS-TR-91-06	___CS-TR-89-06	___5253:TR:87	___5210:TR:86	___5139:TR:84	___5051:TM:82
___CS-TR-91-05	___CS-TR-89-05	___5251:TR:87	___5207:TR:86	___5137:TR:84	___5035:TR:82
___CS-TR-91-04	___CS-TR-89-04	___5250:TR:87	___5205:TR:85	___5136:TR:84	___5034:TR:82
___CS-TR-91-03	___CS-TR-89-03	___5249:TR:87	___5204:TR:85	___5135:TR:84	___5033:TR:82
___CS-TR-91-02	___CS-TR-89-02	___5247:TR:87	___5202:TR:85	___5134:TR:84	___5029:TM:82
___CS-TR-91-01	___CS-TR-89-01	___5246:TR:87	___5200:TR:85	___5133:TR:84	___5018:TM:82
___CS-TR-90-18	___CS-TR-88-22	___5244:TR:87	___5198:TR:85	___5132:TR:84	___5017:TM:82
___CS-TR-90-17	___CS-TR-88-21	___5243:TR:87	___5197:TR:85	___5129:TR:84	___5015:TR:82
___CS-TR-90-16	___CS-TR-88-20	___5242:TR:87	___5195:TR:85	___5128:TM:84	___5014:TR:82
___CS-TR-90-15	___CS-TR-88-19	___5241:TR:87	___5194:TR:85	___5125:TR:84	___5012:TM:82
___CS-TR-90-14	___CS-TR-88-18	___5240:TR:87	___5193:TR:85	___5123:TR:84	___5000:TR:82
___CS-TR-90-13	___CS-TR-88-17	___5239:TR:87	___5190:TR:85	___5122:TR:84	___4684:TR:82
___CS-TR-90-12	___CS-TR-88-16	___5238:TR:87	___5189:TR:85	___5114:TM:84	___4655:TR:81
___CS-TR-90-11	___CS-TR-88-15	___5236:TR:86	___5185:TR:85	___5112:TR:83	___2276:TM:78

Self-timed Mesh-Routing Chips

(Caltech EMRC2-series chips)

Abbreviated, emailable
SPECIFICATIONS

Submicron Systems Architecture Project
Department of Computer Science
California Institute of Technology

30 June 1992 -- Chuck Seitz
8 July 1992 -- minor revisions; added pin characteristics (CS)

1. Introduction

The EMRC2 "Elko Mesh-Routing Chip, version 2" provides packet communication and routing in a 2D-mesh network that connects a set of computing nodes. The EMRC2 is the simplest of the family of "Elko" routing, communication, and interface chips. It may be used in together with Elko router-interface (ERI) chips that interface the EMRC's processor channels to synchronous buses, and with "Slack" chips that translate the EMRC's 0-slack channel protocol to a protocol that maintains high bandwidth over long cables.

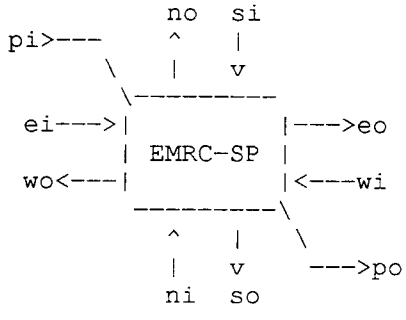
[[2D, 9-bit (EMRC2-2D9) and 1D, 16-bit (1D16) variants of the basic 2D, 8-bit (EMRC2) chip are in development, and will be available in late September 1992.]]

The EMRC2 is the latest in a progression of routing chips that has included the GMRC (Ginzu) routers (1987) used in the Ametek S2010 and MCC ES-kits, the FMRC (Frontier) routers (1989) used in the Intel Delta Touchstone and in several other projects, and an earlier (1991, EMRC1) version of the EMRC router. The Ginzu, Frontier, and Elko routers differ in internal design, layout, speed, and setup-time margins. The EMRC2 is pin-compatible with these earlier routers, but is sufficiently different in setup-time margins that the EMRC2 may not be entirely interchangable with all of these earlier routers.

The EMRC2 router employs oblivious, dimension-order routing that is assured to be free of deadlock so long as consumption at the node output channel is assured. This specification does not go into these technical issues; for a more general discussion of multicomputer message-passing networks, please see section 3 of [Charles L. Seitz, "Multicomputers," chapter 5 in _Developments in Concurrency and Communication_, edited by C.A.R.Hoare, Addison-Wesley, 1991]. Extensive information about the performance of these networks can be found in a series of Caltech Computer Science technical reports by Michael J. Pertel [Caltech-CS-TR-92-4, -5, -6, and -9].

2. Signal names

The EMRC2 chip connects to 10 channels, 5 input and 5 output. The "news" channels -- "n" (northbound), "e" (eastbound), "w" (westbound), and "s" (southbound) -- form the mesh fabric. Note that these compass directions refer to the direction of the packets traversing these channels, not to the side of the chip on which they appear; thus, the east-output (eo) channel appears on the east side of the chip, but the east-input (ei) channel appears on the west side of the chip. The "p" (processor) channels that convey packets to and from the node employ the same self-timed, asynchronous signalling as the "news" channels.



Each of these ten input or output channels is 11 wires, and the 110 signals have 3-character names formed as:

					r	Request
					a	Acknowledge
					t	Tail bit
					0	Data bit 0
North	n				1	Data bit 1
South	s		i		2	Data bit 2
East	e	+	o	+	3	Data bit 3
West	w				4	Data bit 4
Processor	p				5	Data bit 5
					6	Data bit 6
					7	Data bit 7

The first character {n|e|w|s|p} indicates the direction of the packets entering a channel input or leaving a channel output, and the second character {i|o} whether this is a channel input or output. For example, "ni" is an input for northbound (+y) packets. The third character indicates the self-timed request (r) and acknowledge (a) signals, the tail bit (t), and the 8 bits of data (0-7). Each byte transferred is acknowledged for timing and flow control; hence, the bytes are referred to as flits (flow-control units). Individual signals are input or output to the EMRC according to the second character being i or o, except for the acknowledge signals, for which signals ending in "oa" are chip inputs and signals ending in "ia" are chip outputs.

The request and acknowledge signals employ transition signalling, and initialize to LOW in positive logic. The data and tail bits are positive logic.

3. Reset signal

The EMRC-SP has a negative-logic "/ri" reset input intended to be used only for initialization of the entire mesh network. (It is best in CMOS logic for reset signals to be negative logic, so that the reset is asserted while it held at GND during power-up, and is then switched to HIGH after power is established. A positive-logic reset signal cannot be asserted when power is off because the input-protection structures will clamp the input to be less than Vdd, and the reset signal may itself attempt to power the chip.) In order to simplify reset chaining, the reset signal is amplified and provided as the "/ro" output signal.

4. Packet format

A packet may be of any length so long as it includes the appropriate header. A header flit is represented as follows:

bit:	7	6	5	4	3	2	1	0	t
	sign	B	MSB.....delta-X or -Y....LSB						0

The sign bit is 0 for positive and 1 for negative; the delta-X or delta-Y distance is represented in sign and magnitude, with the 6-bit magnitude represented in binary. B specifies packet broadcast, which is not implemented in the current EMRC2; this header bit should be generated as 0 by the interface logic to assure compatibility with future routers that will be capable of broadcast.

On the pi, ei, or wi channels, the header is composed of a delta-X flit followed by a delta-Y flit. The sign of the delta-X flit is significant only for the pi channel: if the magnitude is non-zero, the sign determines whether the packet is routed east (+) or west (-), and the packet will leave the router with the magnitude of the distance decremented. When a packet enters a router on pi, ei, or wi with the magnitude equal to zero, the delta-X header flit is stripped off, and the packet is passed to the Y router.

For packets that enter the Y router from the X router, or that enter the Y router on the ni or si channels, the header is composed only of a delta-Y flit. The Y router is identical to the X router, but connects to northbound (+) and southbound (-) rather than eastbound and westbound channels. When a packet enters the Y router with the delta-Y magnitude equal to zero, the delta-Y header is stripped off, and the packet is routed to the po channel.

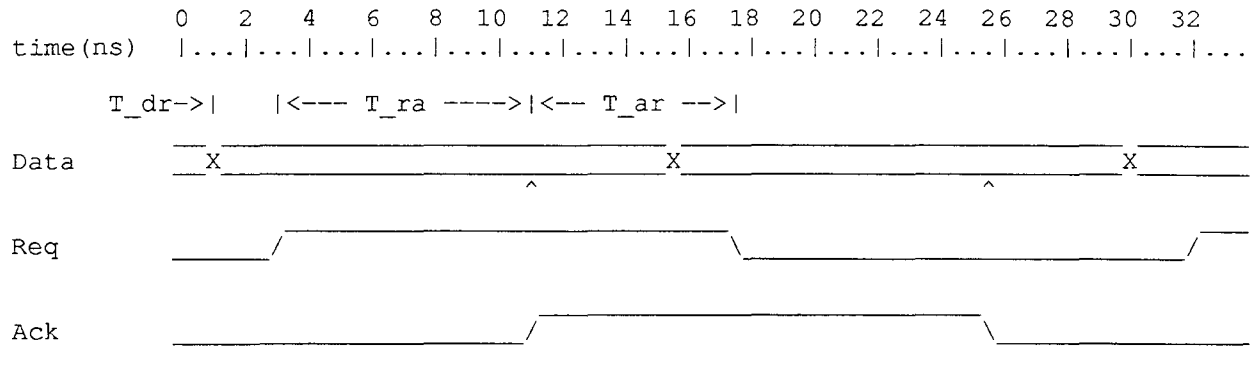
If delta-X = delta-Y = 0 on the pi channel, the packet is routed as usual through the X and Y routers back to the po channel.

The packet is terminated and the path freed by any non-header flit in which the t bit is 1.

5. Timing and performance

The request and acknowledge signals of a channel conform to the pipeline form of 2-cycle (transition) signalling shown in figures 7.16 and 7.24 of [Mead & Conway, _Introduction to VLSI Systems_, Addison-Wesley, 1980]. Each flit is conveyed by a transition of the request signal followed by a transition of the acknowledge signal. This is a "zero-slack" protocol: The channel output may not generate another request until the outstanding request has been acknowledged by the channel input.

At a channel output, data signals preceed the transition of {newsp}or by T_dr, and remain stable until after the corresponding transition of {newsp}oa. At a channel input, data inputs are sampled coincident with the transition of {newsp}ia (T_ad = 0). The point at which data is sampled at a channel input is marked by ^ in the timing diagram below.



In terms of these four parameters (T_dr, T_ra, T_ad, and T_ar), the period is (T_ra + T_ar), the setup-time margin is (T_dr + T_ra + T_ad), and the hold-time margin is (T_ad + T_ar - T_dr). These expressions can be augmented to allow for delay and skew in the wires that connect two routers. Note that the addition of wire delay increases the hold-time margin, but not the setup-time margin; thus, the routers are designed to have a larger setup-time margin than hold-time margin.

The minimum values for the parameters T_dr, T_ra, and T_ar can be observed directly on a channel that is operating at full rate, and T_ad is close to zero. T_ra is determined by the receiving router, and may be larger than its minimum value if the flow is blocked "downstream." T_ar is determined by the sending router, and may be larger than its minimum value if the packet "upstream" is being supplied to the sending router more slowly than the output channel can operate. TYPICAL values for these key parameters at 25C and Vdd = 5V, with routers connected by 4" PCB traces (approximately 20pF, consisting of 15pF wire and 5pF input-pin capacitances) are:

At a channel output,		
{newsp}oa to {newsp}or	T_ar	> 6.5ns
data preceeds {newsp}or by	T_dr	= 2.0ns
At a channel input,		
{newsp}ir to {newsp}ia	T_ra	> 8.0ns
data sampled relative to ack	T_ad	= 0 +/- 0.5ns

(For >, read "equal-to-or-greater-than.") The typical period is thus > 14.5ns (corresponding to < 69MB/s), the setup-time margin is > 10ns +/- 0.5ns, and the hold-time margin is > 4.5ns +/- 0.5ns.

TYPICAL path-formation latency (ei-eo, wi-wo, ni-no, and si-so) for the head of a packet is 30ns.

6. Electrical characteristics

Typical input-pin capacitance is 5pF, and inputs are protected against ESD and latchup to at least +/- 100mA. The input switching threshold at Vdd=5V is 2.4V +/- 0.1V. Timing measurements are between times at which signals cross the switching threshold.

All output pins are driven by a p-channel, n-channel MOSFET pair sized to produce the same transition and delay times for positive-switching and negative-switching signals. Outputs are ESD- and latchup-protected.

The following I-V characteristics of the output and input pins was obtained by electrical measurement of a TYPICAL chip at 25C and Vdd = 5V. The sense of the currents shown is into the pin.

pin voltage (V)	high output (mA)	low output (mA)	input (mA)	
-2.0	<-100.0	<-100.0	<-100.0	
-1.5	<-100.0	<-100.0	-75.10	
-1.0	-66.57	-46.30	-9.945	
-0.5	-35.18	-12.32	-0.0004	
0.0	-34.63	-0.37	0 *	
0.5	-34.00	10.55	0 *	
1.0	-33.21	19.16	0 *	
1.5	-32.14	24.93	0 *	
2.0	-30.60	28.20	0 *	
2.5	-28.36	29.69	0 *	+/- << 1nA
3.0	-25.16	30.24	0 *	
3.5	-20.79	30.47	0 *	
4.0	-15.13	30.57	0 *	
4.5	-8.13	30.63	0 *	
5.0	0.11	30.67	0 *	
5.5	9.44	30.73	0.00006	
6.0	50.19	66.18	11.69	
6.5	>100.0	>100.0	73.92	
7.0	>100.0	>100.0	>100.0	

Although the static saturation current of the p-channel output driver to 0V is ~35mA, whereas the static saturation current of the n-channel driver to Vdd=5V is ~31mA, the devices exhibit somewhat different nonlinearities. The net result of the transistor sizing is to produce very similar (0.2ns difference) switching and delay times into typical (20pF) lumped-capacitive loads. The impedance of a low output close to 0V is ~44ohms, and of a high output close to 5V is ~57ohms.

Power dissipation is determined essentially entirely by the output drivers, and can be calculated based on the load capacitance and expected number of transitions/s. For example, the average number of transitions per flit for random data is 6 (1 request, 1 acknowledge, and 4 data transitions, ignoring the occasional tail), resulting in an average energy of 3nJ/flit if the signals drive 20pF loads (6 * 20pF * (5V)^2 = 3nJ). Thus, if the average, aggregate throughput of a router is 100Mflits/s, the average power is 300mW.

7. Interconnect requirements

Due to conduction overlap and drain-to-gate (Miller) capacitance of the pad drivers, the effective current in the mid-range of switching is ~20mA. When an output drives another input and a short PCB run of ~4", the risetime and falltime are several times longer than the flight time through the wire, and the switching can be characterized by the effective current and the lumped capacitance of the wire and input pin (~20pF). The 20mA current and 20pF capacitance result in a slew rate of approximately 1V/ns, and rise and fall times from the 10% to 90% points of approximately 4ns.

If an output drives a larger capacitance, such as a 12", 50ohm, buried line of 44pF capacitance in a circuit board with e_r = 4.7, the slew rate would be reduced to ~0.4V/ns (20mA driving 44pF of wire + 5pF of input pin), the 10%-90% switching would be increased to ~10ns, and the *additional* delay for the signal to reach the switching threshold would be ~3ns. Together with the ~2.2ns flight time through the wire, the additional ~5.2ns *in each direction* would increase the period of the router by ~70%.

The use of low-impedance (high-capacitance, typically buried) PCB lines is discouraged for distances larger than ~4", particularly for routing chips packaged in the usual 132PGA package. In addition to the impact on performance, the larger capacitance in combination with package-pin inductance may compromise signal integrity and increase dI/dt effects in the Vdd/GND distribution. However, higher impedance (lower capacitance, typically surface) PCB runs up to about 12" are feasible.

When an output signal is not switching, its output impedance (Z) is 40-60ohms. However, while the output is switching, its Z is high -- that of a current source. EMRC2 outputs can, nevertheless, act as a source-terminated transmission-line drivers for long cables. Channels may be connected through up to about 20'-long cables -- typically ribbon cables with alternate wires grounded. Series resistors at the routing-chip outputs may be used to improve the match to the characteristic impedance of the transmission line. Because of the 0-slack protocol, reflections from the far end of *long* cables are assured to return while outputs are not switching. Bandwidth, however, is limited by the round-trip cable delay.

Lines of intermediate lengths, roughly 1' to 2', cannot be treated as lumped-capacitance loads; in addition, the transmission-line reflection may return while an output is still switching, and is not correctly absorbed. Separate transmission-line-driver amplifiers of closely matched delay are recommended in this case.

Slack chips with separate transmission-line-driver chips are recommended for maintaining high bandwidth through long cables.

8. Mesh-edge termination

Request inputs at the mesh edges must be tied to GND. Unused inputs at the mesh edges should be tied to GND in adverse electrical environments. Acknowledge inputs at the mesh edges should be tied to their corresponding request outputs to allow packets to be routed off the edge of the mesh. Otherwise, packets routed off the edge of the mesh will be blocked into the network.